

November/
Dezember 1989

Heft Nr. 6
84037

DM 24,80
ÖS 175,-
sfr 24,80

Microsoft System Journal

DDE • Windows
80386 • 80x87

DDE (Dynamic Data Exchange) hat sich in der Windows-Umgebung als konsistente, flexible Methode für die Kommunikation zwischen Anwendungen bewährt. Bei der Umsetzung auf den Presentation Manager wurden einige Änderungen des Protokolls notwendig, die ausführlich beschrieben werden.

Diskette in jedem Heft

Einfache Zugänge zum PC mit tewi Büchern...

SPRACHEN



C-Gesamtwerk mit Update ANSI C
Herold/Unger, 624 S., **DM 79,–**/sFr 72,70/öS 616,–
Erfolgreicher Kurstext aller C-Konstrukte.
Über 100 lauffähige Beispiele.
ISBN 3-89362-015-X



ANSI C
Herold, 450 S., **DM 79,–**/sFr 72,70/öS 616,–
Programmieren portabler C-Programme im künftigen Sprachstandard. Zahlreiche Musterprogramme. ISBN 3-89362-040-0



Turbo PASCAL 5.0
Ciric/Thies, 496 S., **DM 59,–**/sFr 54,30/öS 460,–
Führt methodisch in alle Aspekte von PASCAL ein. Über 50 kommentierte Musterbeispiele.
ISBN 3-89362-004-4



PC/XT/AT ASSEMBLER-BUCH
Thies, 656 S., **DM 98,–**/sFr 90,20/öS 764,–
Als CPU-Befehle und Makro-Assembler für Programmierer und Systemsoftware-Entwickler.
ISBN 3-921803-88-8



PC/XT/AT Numerik-BUCH
Thies, 650 S., **DM 98,–**/sFr 90,20/öS
Hochgenaue Gleitpunkt-Arithmetik mi
8087/287/387 in Assembler und C. IE
theksfunktionen. ISBN 3-89

BETRIEBSSYSTEME–OBERFLÄCHEN



OS/2: Einführung + Referenz
Beam, 480 S., **DM 79,–**/sFr 72,70/öS 616,–
Führt in 65 Modulen kursartig durch OS/2 und ist zugleich ein Befehlslexikon. ISBN 3-89362-011-7



MS DOS 4.0/PC DOS 4.0: Einführung + Referenz
Stultz, 424 S., **DM 69,–**/sFr 63,50/öS 538,–
Alle DOS-Befehle der letzten Version 4.0 in Form von 57 Modulen. Kurstext und Befehlslexikon.
ISBN 3-89362-010-9



WINDOWS 2.0: Einführung + Referenz
Whitsitt/Bryan, 496 S., **DM 79,–**/sFr 72,70/öS 616,–
Kurstext und Lexikon. Auch für WINDOWS 386 geeignet. ISBN 3-921803-81-0



DESQVIEW
Grieser, 328 S., **DM 59,–**/sFr 54,30/öS 460,–
Beste WINDOWS-Alternative, problemlos für alle PC, XT, AT, hier exzellent dargestellt.
ISBN 3-89362-002-8

PROZESSOREN – UTILITIES



Der 80286-Prozessor
Thies, 320 S., **DM 79,–**/sFr 72,70/öS 616,–
Zeigt exakt und anschaulich Funktion und Programmierung der CPU 80286. Ein Expertentext
ISBN 3-921803-63-2



Der 80386-Prozessor
Thies, 556 S., **DM 89,–**/sFr 81,90/öS 694,–
Keine Befehlssammlung! Zeigt 80386-Philosophie, Hardware, Programmierung für reale Anwender. ISBN 3-921803-68-3



M68000-Familie (Hilf/Nausch) Teil 1 + Teil 2
Teil 1: 576 S., **DM 79,–**/sFr 72,70/öS 616,–
Architektur + Programmierung;
ISBN 3-921803-16-0;
Teil 2: 400 S., **DM 69,–**/sFr 63,50/öS 538,–
Bausteine + Systeme ISBN 3-921803-30-6



PC Tools Deluxe 5.5
Patschorke, 350 S., **DM 59,–**/sFr 54,30/öS 460,–
Norton's erfolgreichster Konkurrent – hier nach Alltagssituationen gegliedert.
ISBN 3-89362-038-9



I DIR PLUS
Walsh, 180 S., **DM 49,–**/sFr 45,10/öS 382,–
Verwaltungsoberfläche + Pannenhilfe für die tägliche PC-Benutzung. Top-Utility in den USA
ISBN 3-89362-041-9

cher erhalten Sie bei Ihrem Buchhändler, in
ter-Fachgeschäften oder in den Fachabteilungen
renhäuser.
ungen: Hotline 0 21 91-34 20 77



tewi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

CH: M+T Vertriebs AG Kollerstr. 3, CH-6300 Zug
A: M+T Verlag GesmbH, Gr.-Neugasse 28, A-1040 Wien
R. Lechner+Sohn, Heizwerkstr. 10, A-1232 Wien



Editorial

Systems 1989: Besucherrekord im Bayerischen High-Tech-Mekka! Erste EISA-Rechner zu sehen! 486er Computer auf breiter Front! Neue Personalcomputer führender Hersteller mit integriertem CD-ROM-Laufwerk!

Das werden die Schlagzeilen in der Computer- und Tagespresse im Oktober und November sein. Aber auch: Neue Netzwerktechnologien vorgestellt! Erste große OS/2-Anwendungen! Der Standard Windows hat sich durchgesetzt! Das sind zweifellos die Software-Highlights dieser Messe.

Microsoft wird neben seinem eigenen Messestand (Halle 5, A8/B5) mit den üblichen Vorführungen neuer Produkte unabhängigen Softwarehäusern einen eigenen Ausstellungspavillion (vor Halle 5) zur Vorstellung neuer Windows- und Presentation Manager-Anwendungen zur Verfügung stellen. Im Microsoft-Zelt (vor Halle 16) werden darüber hinaus zahlreiche Softwarefirmen weitere Windows- und OS/2-Produkte der Öffentlichkeit präsentieren. Dabei reicht die Spanne der beteiligten Unternehmen von Riesen wie zum Beispiel Siemens bis zu kleineren High-Tech-Firmen wie Format Software, die bislang ausschließlich im Macintosh-Markt zu Hause waren.

104 Seiten umfaßt die erste Ausgabe des »Microsoft Windows Applikationsverzeichnisses«, des ersten Katalogs, der sich bemüht, sämtliche in Deutschland derzeit erhältlichen Windows-Anwendungen in Kurzbeschreibungen aufzuführen. Ende August bei Microsoft im Selbstverlag erschienen, wären schon wieder Ergänzungen nötig – so rasant entwickelt sich derzeit der Windows-Markt. Alle halbe Jahre soll deshalb künftig eine Neuauflage erscheinen. Der Katalog ist bei Microsoft gegen einen Unkostenbeitrag von 6,- DM zuzüglich Versandkosten erhältlich.

Das erste Heft des *Microsoft System Journals* in neuer Aufmachung ist bei den Lesern offensichtlich gut angekommen. Die Zahl von 500 neuen Abonnenten in nur vier Wochen spricht für sich. Auch das Feedback auf unsere Leserbefragung war beträchtlich. Über die Einzelergebnisse werden wir im nächsten Heft berichten. Auch die Gewinner der zwanzig QuickPascal-Compiler und der 35 Bücher werden wir dann bekanntgeben. Zehn Produkte wurden verlost, weitere zehn Produkte waren ja den ersten Einsendern des Fragebogens versprochen. Um zu den ersten zehn Rücksendern zu zählen, mußte man sich aber schon wirklich beeilen. Ein Leser schickte deshalb am Erstverkaufstag seine Schwester mit ausgefülltem Bogen direkt in die Redaktionsräume. Zwei andere übersandten uns ihre Antworten per Fax.

Zahlreiche Anfragen gingen mit der Bitte um 3 1/2-Zoll-Listing-Disketten ein. Da wir den Heften nicht Disketten mit beiden Formaten beilegen können, 3 1/2-Zoll-Floppys auch beträchtlich teurer sind als jene im 5 1/4-Zoll-Standard, bieten wir allen Interessenten künftig die Listings gegen eine Aufwandsentschädigung auch auf 3 1/2-Zoll-Disketten an. Zu beziehen über: *Microsoft System Journal*, Leserservice 7311, Postfach 6740, D-8700 Würzburg.

Anregungen, Beiträge, Tips, Kritik, Lob und Tadel für unser *System Journal* sind Microsoft und der Redaktion natürlich stets willkommen. Schließlich ist es IHR Heft, mit dem wir Ihnen wieder viel Vergnügen wünschen!

Dr. Michael Kausch

Pressesprecher
der Microsoft GmbH

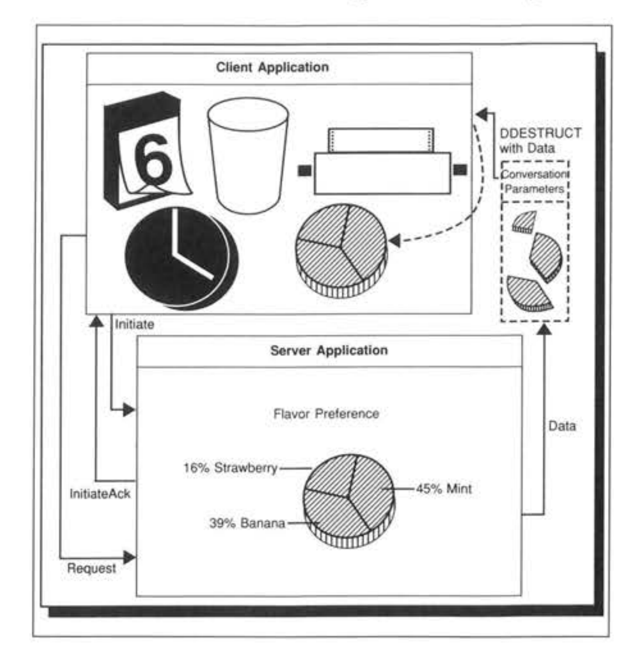
Inhalt

DDE und Presentation Manager:

Dynamischer Datenaustausch unter dem Presentation Manager

IBM und Microsoft lieferten vor einiger Zeit eine neue Version des Betriebssystems OS/2 aus, das einige wichtige Neuerungen gegenüber der ersten Version enthält. Die wichtigste Änderung ist der Presentation Manager (PM), der jetzt zum Lieferumfang gehört. Der Presentation Manager basiert auf Microsoft Windows, und bietet dieselben Vorteile, die Windows unter DOS bietet: eine fensterorientierte, grafische Benutzeroberfläche und die Unterstützung zahlreicher Ein- und Ausgabegeräte.

Eine wichtige Komponente von Microsoft Windows ist auch im Presentation Manager implementiert, das dynamische Datenaustausch-Protokoll (DDE, Dynamic Data Exchange). DDE ist ein veröffentlichtes Meldungsprotokoll für den Austausch von Daten unter den beteiligten Programmen und ist unter Windows schnell als Standardprotokoll für den Datenaustausch anerkannt worden. Der Übergang des DDE von DOS auf OS/2 erforderte einige Verbesserungen bezüglich der Adreßbeschränkung der originalen Windows DDE-Spezifikation. Dieser Artikel beschreibt diesen Übergang und verwendet ein grafisches Datenaustausch-Programm als Beispiel.



▸▸ Bild 1: Im DDE Ein-Server/Ein-Client-Modell initiiert die Client-Anwendung die Konversation und der Server bestätigt sie. Anschließend schickt der Server Daten, wenn sie der Client anfordert.

Protokolländerungen

In der Windows-Umgebung bietet DDE eine konsistente, flexible Methode für die Konversation zwischen Anwendungen. Bei der Umsetzung auf die geschützte Multitasking-Umgebung des PM wurden jedoch einige Änderungen des Protokolls notwendig. Diese Änderungen sollten die neuen Gegebenheiten der OS/2-Umgebung berücksichtigen, ohne das DDE-Modell, das sich in der Windows-Umgebung bewährt hat, sehr zu ändern.

Die erste Annäherung an OS/2 und den Presentation Manager wurde durch eine einfache Änderung der Meldungsparameter erreicht. Die wichtigste Änderung ist dann notwendig, wenn Daten an eine andere Anwendung übergeben werden, was unter OS/2 prozeßübergreifend ist. Wo unter Windows eine Handle auf die Daten genügt, wird für die Datenübergabe unter OS/2 ein Speicherselektor benötigt. String-Daten könnten weiterhin in der globalen Atomtabelle übergeben werden, die Anwendungen müßten aber den Zugriff auf diese Atomtabelle explizit anfordern. Diese Annäherung löste das Problem, das durch den geschützten Speicher für die DDE-

Nachrichten notwendig wurde. Doch es gab noch weitere Probleme und Einschränkungen. Sie konnten aber durch weitere Änderungen des Protokolls gelöst werden.

Ein Problem für DDE stellt die Beschränkung auf zwei Parameter in den PM-Nachrichten dar. Analog zu dem DDE-Modell von Windows ist der erste Parameter in der DDE-Nachricht die Handle des sendenden Fensters. Es bleibt also nur noch ein 32-Bit-Parameter für die anderen Konversationsparameter und die Datenreferenzen übrig. Obwohl die nötigen Parameter und die Datenreferenzen in den übrigbleibenden Long-Wert passen würden, wäre dann aber kein Platz für Erweiterungen geblieben.

Die Konversation mit anderen Computern in einem lokalen Netz (LAN) oder mit anderen Typen von Computern bereitet mit dieser Parametereinschränkung Schwierigkeiten. Die Einschränkung wird auch zum Problem, wenn sich der adressierbare Speicher des Betriebssystems vergrößert. Selbst in der jetzigen Umgebung ist jegliche Erweiterung des DDE-Modells oder der Konversationsparameter wegen der beschränkten Parameteranzahl nicht möglich. Das Protokoll mußte also einfach so erweitert werden, daß es Raum für Neuerungen bietet und der Anwendung ermöglicht, eventuell notwendige eigene Parameter unterzubringen.

Zur Erweiterung des Parameterbereichs für die DDE-Nachrichten verwendet die PM-Version des DDE den zweiten Parameter als einen 32-Bit-Zeiger auf eine von zwei verfügbaren DDE-Strukturen. Diese Strukturen beinhalten sowohl alle nötigen DDE-Konversationsparameter, als auch – falls notwendig – die tatsächlichen Daten. Wenn die Anforderungen an DDE sich ändern, so kann diese Struktur erweitert werden, ohne die Parameter der DDE-Nachricht zu ändern. Die Far-Adresse der Struktur bietet Kompatibilität zu neuer Systemsoftware und anderen Maschinen. Durch das Packen aller Parameter in eine Struktur werden die Nachrichtenparameter konsistenter, da unterschiedliche Parameter in der Struktur selbst enthalten sind.

Alle Parameter werden in eine einzige Struktur gepackt, wodurch die Verwendung der globalen Atomtabelle nicht mehr notwendig ist. Das Hinzufügen von Stringparametern zu der globalen Atomtabelle hätte zu einem zweiten Datenzugriffspfad geführt, der das Protokoll nur unnötig verkompliziert hätte. Statt dessen sind die String-Daten nun in der DDE-Struktur enthalten.

Da die meisten DDE-Nachrichten an Fenster in einem anderen Prozeß gesendet bzw. übergeben werden, muß der Speicher der DDE-Strukturen dem empfangenden Prozeß zugänglich gemacht werden. Das PM-DDE-Protokoll bietet ein neues Anwendungs-Programm-Interface (API) für das Senden und Empfangen von DDE-Nachrichten. Die Anwendungen benutzen nicht WinPostMsg oder WinSendMsg für die Übertragung der DDE-Nachrichten. Statt dessen werden die Nachrichten und die Parameter an ein System-API übergeben, das es dem empfangenden Prozeß gestattet, auf die DDE-Strukturen zuzugreifen und die Meldungen im Namen der aufrufenden Anwendung schickt oder ablegt. Diese APIs gewährleisten den konsistenten und korrekten Zugriff auf die Strukturen und vermindert den Programmieraufwand, der für eine Anwendung, die DDE verwendet, notwendig ist.

Diese Verbesserungen von DDE unter OS/2 bieten einen Standardrahmen für kommunizierende Anwendungen, ohne daß es notwendig ist, ein neues Protokoll zu entwickeln, das DDE in einer Multitasking-Umgebung gerecht wird. Zusätzlich bietet DDE für OS/2 eine einfache Methode, um sogar den immer häufiger werdenden komplexen grafischen Datenaustausch in einer effizienten Weise zu erledigen.

Ein Client, ein Server

Der einfachste Fall für den Einsatz von DDE besteht aus einer Anwendung, auch Client genannt, die von einer anderen unabhängigen Anwendung (dem Server) Daten benötigt. Ein klassisches Beispiel ist ein Geschäftsgrafik-Programm, das aktualisierte Daten von einem Tabellenkalkulations-Programm empfängt, und diese geänderte Daten durch Neuausgabe eines Diagramms widergibt. In diesem Fall ist das Geschäftsgrafik-Programm der Client und die Tabellenkalkulation der Server.

Wir wollen uns zunächst mit dem einfachsten Fall des Ein-Client-/Ein-Server-Modells beschäftigen. In diesem Beispiel ist der Zweck der DDE-Konversation der Austausch von grafischen Daten zwischen unterschiedlichen Anwendungen. Die Server-Anwendung ist irgendein Programm, das im Fenster der Client-Anwendung Daten grafisch darstellen will. Die grafischen Daten werden immer dann, wenn die Server-Anwendung das Aussehen seines Bildes ändern will, gesammelt und an die Client-Anwendung gesendet.

Bild 1 zeigt den generellen Programmablauf für eine Ein-Client-/Ein-Server-Anwendung. Die Client-Anwendung beginnt die Konversation. Nachdem der Server den Beginn bestätigt hat, fordert der Client die Daten an, und der Server sendet sie in den entsprechenden Strukturen.

Eine Ein-Client-/Ein-Server-DDE-Konversation beginnt, wenn die Client-Anwendung eine WM_DDE_INITIATE-Nachricht an alle anderen obersten Fenster im System gesendet hat. Der Client gibt den Namen der Anwendung an, die antworten soll, ebenso den Namen, der das Thema der gewünschten Konversation angibt. Jeder der beiden Strings kann NULL sein, um anzuzeigen, daß der Server oder das Thema der Konversation nicht festgelegt ist, und jede An-

```
case WM_CREATE:    /* broadcast the initiate message  */
   WinDdeInitiate(hwnd, "App_Name", "Graphics_Exchange");
   break;
```

◂◂ Listing 1: Initiierung der DDE-Konversation.

```
case WM_DDE_INITIATE:  /* respond if app and topic strings match */
 pDDEInit = (PDDEINIT)lParam2;
 if ((HWND)lParam1 != hwnd) {
    if((!strcmp("App_Name", pDDEInit->pszAppName)) &&
        (!strcmp("Graphics_Exchange", pDDEInit->pszTopic))) {
           WinDdeRespond(lParam1, hwnd, "Client",
           "Graphics_Exchange");
    }
 }
 DosFreeSeg(PDDEITOSEL(pDDEInit));
break;
```

◂◂ Listing 2: Antwort auf eine DDE-Konversation.

wendung, die DDE implementiert hat, mitmachen kann. Die Nachricht WM_DDE_INITIATE wird nicht von der Anwendung direkt gesendet. Statt dessen verwendet die Client-Anwendung die Funktion WinDdeInitiate, um die Nachricht zu senden. *Listing 1* zeigt die Prozedur für die Initiierung.

Erhält eine Server-Anwendung die Nachricht WM_DDE_INITIATE, so prüft sie den Namen der Anwendung und die Art der Konversation, um zu entscheiden, ob sie teilnehmen will. Ein Beispiel für den Beginn der Bearbeitung sehen Sie in *Listing 2*. Beachten Sie, daß die Stringparameter der Funktion WinDdeInitiate nicht als Parameter in der Nachricht WM_DDE_INITIATE enthalten sind. Statt dessen befinden sie sich in der Struktur DDEINIT, auf die durch den zweiten Parameter verwiesen wird. In allen DDE-Nachrichten enthält der erste Parameter die Handle des Fensters, von dem die DDE-Nachricht stammt.

Wenn der Server an der Konversation teilnehmen möchte, ruft er die Funktion WinDdeRespond auf, um die Nachricht WM_DDE_INITIATEACK an die Client-Anwendung zurückzusenden. Auch hier werden die Stringparameter in die DDEINIT-Struktur kopiert, und der Zeiger auf die Struktur wird als zweiter Parameter in der Nachricht WM_DDE_INITIATEACK übergeben.

Nach dem Verbindungsaufbau kann der eigentliche Datenaustausch beginnen. Dieser Austausch kann entweder einmalig oder fortlaufend sein, oder auf Kommandos der Gegenseite beruhen.

Der einmalige Datentransfer wird mit der Nachricht WM_DDE_REQUEST durchgeführt, wenn die Daten vom Server zum Client übertragen, und durch die Nachricht WM_DDE_POKE, wenn die Daten vom Client zum Server übertragen werden. In beiden Fällen wird eine DDESTRUCT-Struktur von der Client-Anwendung allokiert und gefüllt. Die Struktur enthält Statusflags, die das Format der Daten, die übertragen oder angefordert werden, beschreiben, wie auch den Zustand und die Größe der Daten. Nachdem die Struktur gefüllt ist, wird die Nachricht WM_DDE_REQUEST oder WM_DDE_POKE mit der Funktion WinDdePostMsg an die Server-

```
case WM_DDE_REQUEST: /* allocate DDESTRUCT and send data */
  pDDEStruct = (PDDESTRUCT)lParam2;
  strcpy(szTemp, "Text_Data");
  if((pDDEStruct->usFormat == DDEFMT_TEXT) &&
     (!strcmp(szTemp, DDES_PSZITEMNAME(pDDEStruct)))) {
         nNumBytes = (strlen(szTemp) + 2 + strlen(szData));
         pDDEStruct = st_DDE_Alloc((sizeof(DDESTRUCT) + nNumBytes),
                                  "DDEFMT_TEXT");
         pDDEStruct->cbData = strlen(szData) + 1;
         pDDEStruct->offszItemName = (USHORT)sizeof(DDESTRUCT);
         pDDEStruct->offabData = (USHORT)((sizeof(DDESTRUCT) +
                                   strlen(szTemp)) + 1);
         memcpy(DDES_PSZITEMNAME(pDDEStruct), szTemp, (strlen(szTemp)
              + 1));
         memcpy(DDES_PABDATA(pDDEStruct), szData, (strlen(szData)
              + 1));
         pDDEStruct->fsStatus |= DDE_FRESPONSE;
         WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_DATA, pDDEStruct,
                  TRUE);
         DosFreeSeg(PDDESTOSEL(lParam2));
}
else {  /* send negative ACK using their DDESTRUCT */
     pDDEStruct->fsStatus &= (~DDE_FACK);
     WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_ACK, pDDEStruct,
                 TRUE); }
break;
case WM_DDE_POKE: /* unsolicited data from client */
  pDDEStruct = (PDDESTRUCT)lParam2;
  strcpy(szTemp, "Text_Data");
  if((pDDEStruct->usFormat == DDEFMT_TEXT) &&
     (!strcmp(szTemp, DDES_PSZITEMNAME(pDDEStruct)))) {
          strcpy(szData, DDES_PABDATA(pDDEStruct));
          DosFreeSeg(PDDESTOSEL(lParam2)); }
else {  /* send negative ACK using their DDESTRUCT */
     pDDEStruct->fsStatus &= (~DDE_FACK);
     WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_ACK, pDDEStruct,
                 TRUE); }break
```

◂ Listing 3: Server-Programmteil für eine einmalige Datenübertragung.

Anwendung übertragen. Dieser Aufruf wird für alle Meldungen bis auf WM_DDE_INITIATE und WM_DDE_INITIATEACK verwendet. Der Aufruf stellt sicher, daß die empfangende Fenster-Handle Zugriff auf den für DDESTRUCT allokierten Speicher erhält.

Wenn die Daten gesendet werden, wird für den Transfer nur die Nachricht WM_DDE_POKE benötigt, da die Übertragung unaufgefordert stattfindet. Werden die Daten angefordert, so antwortet die Server-Anwendung mit der Nachricht WM_DDE_DATA. Kann der Server die Daten nicht im gewünschten Format liefern, so antwortet er mit einer negativen WM_DDE_ACK-Nachricht. *Listing 3* zeigt den Programmteil, der in der Server-Anwendung notwendig ist, um beide einmaligen Datentransfermethoden zu implementieren.

Wahrscheinlich ist die am meisten benötigte Datentransferart in DDE die kontinuierliche Verbindung zwischen Anwendungen. In diesem Fall sendet die Client-Anwendung eine WM_DDE_-ADVISE-Nachricht. Die Struktur DDESTRUCT wird in derselben Art und Weise wie bei der Nachricht WM_DDE_REQUEST gefüllt. Diese Nachricht informiert die Server-Anwendung, daß die Client-Anwendung eine WM_DDE_DATA-Nachricht erhalten möchte, wenn sich Daten ändern. Wie bei WM_DDE_REQUEST antwortet der Server mit einer negativen WM_DDE_ACK-Nachricht, wenn er die Daten nicht im geforderten Format liefern kann. Können die Daten geliefert werden, so erhält die Client-Anwendung weiterhin WM_DDE_DATA-Nachrichten, bis entweder der Austausch beendet ist oder die ständige Verbindung gelöst wird. *Listing 4* zeigt die Behandlung einer permanenten Verbindung durch den Server.

► *Listing 4: Server-Programmteil für eine ständige Datenverbindung.*

```
case WM_DDE_ADVISE:  /* set ADVISE bit in window data and ACK */
 pDDEStruct = (PDDESTRUCT)lParam2;
 if(pDDEStruct->usFormat == DDEFMT_TEXT) {
    pWWVar->bAdvise = TRUE;
    if((pDDEStruct->fsStatus & DDE_FACKREQ) == DDE_FACKREQ) {
       pDDEStruct->fsStatus |= DDE_FACK;
       WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_ACK, pDDEStruct,
                     TRUE);
    }
    else {
        DosFreeSeg(PDDESTOSEL(lParam2));
    }
 else {    /* Send a negative ACK using their DDESTRUCT */
    pDDEStruct->fsStatus &= (~DDE_FACK);
    WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_ACK, pDDEStruct, TRUE);
 }
break;
```

► *Listing 5: Server-Programmteil für die entfernte Ausführung von Kommandos.*

```
case WM_DDE_EXECUTE:                       /* execute a command */
   pDDEStruct = (PDDESTRUCT)lParam2;
   strcpy(szCommand, DDES_PABDATA(pDDEStruct);
   if(!Dde_Cmd_Processor(szCommand)) {   /* parse and execute
                                            the command */
         pDDEStruct->fsStatus &= (~DDE_FACK);
         WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_ACK,
                       pDDEStruct, TRUE);
 }
 else {
    DosFreeSeg(PDDESTOSEL(lParam2));
 }
break;
```

Beendet der Client die ständige Verbindung, so sendet er die Nachricht WM_DDE_UNADVISE. Diese Nachricht beendet nicht die DDE-Konversation, aber es werden bei Datenänderungen keine WM_DDE_DATA-Nachrichten mehr geschickt.

Eine weitere Notwendigkeit einer DDE-Konversation ist das Ausführen von Kommandos eines Servers auf Anweisung des Clients. In diesem speziellen Fall enthält die Nachricht DDESTRUCT einen String der Kommandos, die ausgeführt werden sollen. An den Client wird dann je nach Ergebnis der Ausführung eine positive oder negative WM_DDE_ACK-Nachricht gesendet. Ein Beispiel für die entfernte Ausführung von Befehlen sehen Sie in *Listing 5*.

Das Ende der DDE-Konversation kann durch den Client oder den Server veranlaßt werden. Normalerweise wird die Nachricht WM_DDE_TERMINATE gesendet, wenn der Anwender die Anwendung beendet, was aber nicht immer der Fall ist. Was auch immer der Grund für die Beendigung ist, das Senden der Nachricht WM_DDE_TERMINATE veranlaßt, daß keine weiteren DDE-Nachrichten gesendet werden. Die Anwendung, die die Nachricht WM_DDE_TERMINATE sendet, darf sich nicht beenden, bis die andere Anwendung mit einer WM_DDE_TERMINATE-Nachricht geantwortet hat. Bei der Beendigungs-Nachricht werden keine Parameter benötigt.

Benutzerdefinierte Formate

Eine Anwendung kann eigene Datenformate erzeugen, wenn kein geeignetes Systemformat verfügbar ist. Das System bietet ein vordefiniertes Format für den Austausch von Textstrings, DDEFMT_TEXT. Bevor eine Anwendung ein selbstdefiniertes Datenformat benutzt, muß es

►► *Listing 6: Registrierung eines Benutzer-DDE-Datenformats.*

```
USHORT Register_DDEFMT(pszFormat)
PSZ pszFormat;
{
HATOMTBL hAtomtbl;
USHORT   retn;

   hAtomtbl = WinQuerySystemAtomTable();
   if (retn = WinFindAtom(hAtomtbl,pszFormat))
      return retn;
   else
      return (WinAddAtom(hAtomtbl,pszFormat));
   }
```

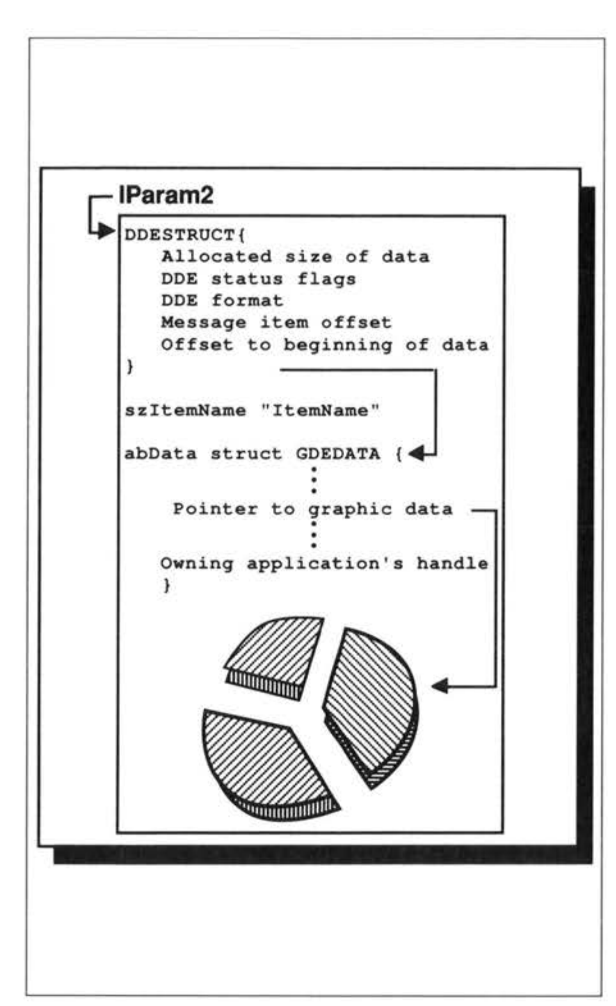
►► *Listing 7: Allokierung von gemeinsamem Speicher für die DDE-Meldungskonversation.*

```
      /* send a request for data */
      /* allocate memory  */
      DDEstrptr = DDE_Alloc(sizeof(DDESTRUCT), IDS_GDE);
      WinDdePostMsg((HWND)lParam1, DDEtoHWND, (ULONG)WM_DDE_REQUEST,
              DDEstrptr, TRUE);...PDDESTRUCT DDE_Alloc(size, format)
 int size;
 char *format;
 /*************************************************************
 *  1. Allocate a block of size. bytes
 *     of giveable, shared memory for DDE call.
 *  2. Fill in DDE data format by
 *     calling Register_DDEFMT((PSZ)format);.
 *************************************************************/
{
SEL     ddepsel;
USHORT  dasret;
PDDESTRUCT DDEstrptr;
  if ((dasret = DosAllocSeg(size, &ddepsel, SEG_GIVEABLE)) == 0) {
      DDEstrptr = (PDDESTRUCT)SELTOPDDES(ddepsel);
      memset(DDEstrptr, (BYTE)NULL, size); /*  set allocated memory
                                               to nulls  */
      /* fill in DDE data format */
       DDEstrptr->usFormat = Register_DDEFMT((PSZ)format);
   } else {   /*  error  */
      return((PDDESTRUCT)NULL);
    }
}
```

eine eindeutige Identifikation anfordern und dieses Format registrieren, so daß andere Anwendungen diese Format-Identifikation in der DDESTRUCT-Struktur dem speziellen Format zuordnen können. Obwohl die DDE-Formate denen des Clipboards zu gleichen scheinen, dürfen sie aber nicht mit diesen verwechselt werden. Clipboard-Formate identifizieren Handle-Typen, DDE-Formate hingegen identifizieren das tatsächliche Layout der Daten im DDESTRUCT-Block. Wir haben das Eintragen der DDE-Formate über die System-Atomtabelle gelöst. Das Präfix für DDE-Formate lautet DDEFMT_. Die Verwendung des Atommanagers für die Registrierung von DDE-Formaten garantiert eindeutige IDs in allen Anwendungen, die diese Methode der Formatregistrierung benutzen.

Listing 6 zeigt eine Funktion mit Namen Register_DDEFMT, die demonstriert, wie man ein vom Benutzer definiertes Format im System registrieren kann. Register_DDEFMT liefert das DDE-Format entweder eines bereits existierenden oder eines neu erzeugten Datenformats. Beim ersten Aufruf von Register_DDEFMT mit einem speziellen DDE-Datenformat wird dieses Format in der System-Atomtabelle eingetragen. Jeder Aufruf von Register_DDEFMT mit einem speziellen DDE-Formatstring führt zur Suche dieses Formats und der Übergabe an den Aufrufer. *Bild 2* zeigt einen Beispielaufruf von Register_DDEFMT.

Das benutzereigene DDE-Datenformat für das Beispielprogramm, das grafische Daten austauscht, ist im *Bild 2* dargestellt. Das Format besteht aus einem grafischen Beschreibungsblock



◄◄ *Bild 2: Das Benutzerformat für das DDE-Grafikformat des Beispielprogramms. Pfeile zeigen die Offsets auf die Daten.*

```
MRESULT APIENTRY MasterDDEWndProc(hwnd,message,lParam1,lParam2)
.
.
.
    DDEstrptr = (PDDESTRUCT)lParam2;

    case WM_DDE_DATA:
.
.
.
    DosFreeSeg(PDDESTOSEL(DDEstrptr));
```

◄ *Listing 8: Freigabe des gemeinsamen DDE-Speichers.*

im abData-Bereich. Dieser Kontrollblock für die grafische Beschreibung enthält unter anderem einen Offset-Zeiger auf die grafischen Daten, die im selben gemeinsamen Speicherbereich hinter den DDE-Daten in abData übergeben werden.

Die Verwendung von gemeinsamem Speicher

DDE benutzt gemeinsamen Speicher für die komplette Konversation. Zwei verschiedene Typen von Speicherobjekten werden für die DDE-Konversation allokiert – die Strukturen DDEINIT und DDESTRUCT. Die Art, wie diese Objekte allokiert werden, kann unterschiedlich sein, aber beide führen dazu, daß dem Empfänger gemeinsamer Speicher zugänglich gemacht wird. Die gemeinsamen Speicherobjekte müssen korrekt freigegeben werden, damit die OS/2-Speicherverwaltung für gemeinsamen Speicher richtig funktioniert. Die Datenbereiche, die szAppName und szTopicName bei den Aufrufen von WinDdeInitiate und WinDdeRespond enthalten, können im privaten oder gemeinsamen Speicher liegen. Beim Ausführen der Funktionen WinDdeInitiate und WinDdeRespond kopiert das System diese Strings in DDEINIT und macht DDEINIT dem Empfänger zugänglich.

Der Datenbereich, der die Struktur DDESTRUCT beinhaltet muß beim Aufruf von DosAllocSeg oder DosAllocHuge mit dem Flag SEG_GIVEABLE versehen werden. WinDdePostMsg macht das DDESTRUCT-Speicherobjekt dem Empfänger zugänglich und gibt sie beim Sender frei.

Alle Zeiger, die in abData enthalten sind, müssen auf gemeinsamen Speicher zeigen. Es liegt in der Verantwortung der Anwendung, die Allokierung und Synchronisation dieser gemeinsamen Speicherbereiche vorzunehmen. Die Anwendungen können entweder die Basis-Speicherverwaltungsfunktionen von OS/2 oder höhere Funktionen verwenden, um dies zu erreichen – der Speicher muß nur richtig verwaltet werden.

Der Empfänger der Nachricht muß nach Holen der Informationen alle Speicherobjekte freigeben. Dazu wird DosFreeSeg benutzt. Die Strukturen DDEINIT und DDESTRUCT müssen vom Empfänger der Nachricht freigegeben werden.

Die *Listings 7* und *8* zeigen das Allokieren und Freigeben der gemeinsamen Speicherobjekte, die notwendig sind, um WinDdePostMsg aufzurufen. Der erste Teil von *Listings 7* ist der Aufruf einer lokalen Funktion, die das gemeinsame DDE-Speicherobjekt allokiert und initialisiert. Die Funktion DDE_Alloc() führt die eigentliche Allokierung des gemeinsamen Speichers mit einem Aufruf von DosAllocSeg() durch und löscht das gesamte Speicherobjekt. Der gemeinsame DDE-Speicherblock wird mit dem selbstdefinierten Datenformat durch Setzen des Feldes usFormat in DDESTRUCT initialisiert. Dies erledigt die Funktion Register_DDEFMT(), die gerade erläutert wurde. Durch Allokieren und Initialisieren der gemeinsamen DDE-Speicherobjekte in dieser Weise kann man auf einfache Art gebrauchsfertige gemeinsame Speicherobjekte für alle unsere DDE-Konversationen holen.

Listing 8 zeigt die Deallokierung von DDESTRUCT im Falle der Bearbeitung einer WM_DDE_DATA-Nachricht. Die eigentliche Aufgabe erledigt ein Aufruf der Funktion DosFreeSeg mit dem Selektor des DDE-Speicherobjekts als Parameter.

Die Strukturen DDEINIT und DDESTRUCT

Der Presentation Manager bearbeitet die Daten und allokiert den Speicher für die API-Aufrufe WinDdeInitiate und WinDdeRespond wie Sie es

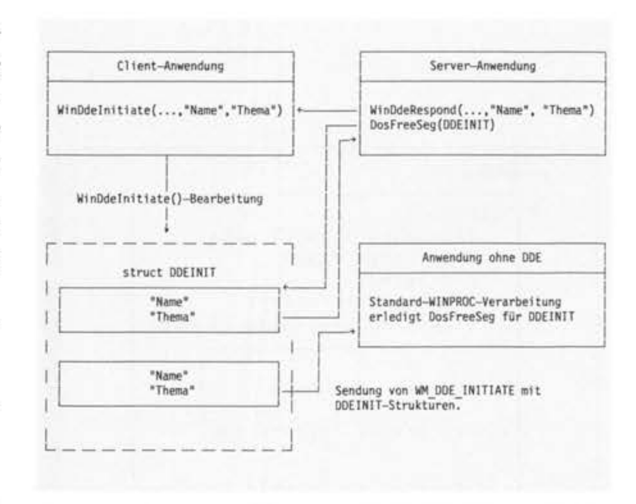
▶ *Tabelle 1: OS/2 DDE-Nachrichten.*

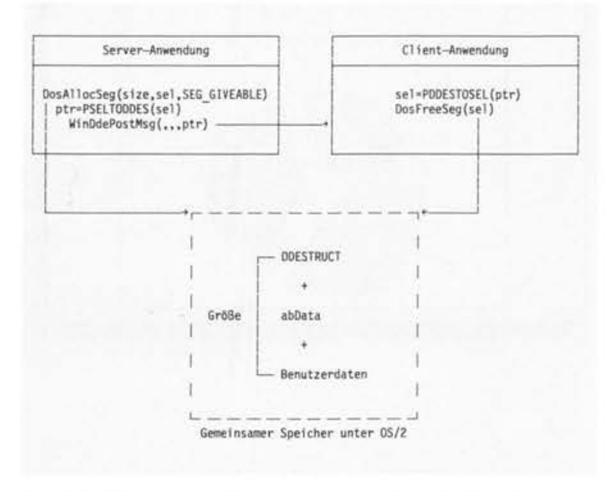
▶▶ *Bild 3: Das System nimmt den bei WinDde-Initiate übergebenen String und kopiert ihn in mehrere DDEINIT-Blöcke, die dann an die Applikationen im System übertragen werden.*

▶▶ *Bild 4: Die Struktur DDESTRUCT wird von der Server-Anwendung allokiert, vom Server an den Client übergeben und dann von der Client-Anwendung freigegeben.*

Jede DDE-Nachricht hat zwei Parameter. Der erste Parameter (das Long-Wort lParam1) enthält die Handle des Senderfensters. Es ist immer dasselbe und wird unten nicht dargestellt. Der zweite Parameter (das Long-Wort lParam2) enthält den Far-Zeiger auf die Struktur DDESTRUCT für WinDDEPostMsg() oder DDEINIT für den Aufruf von WinDDEInitiate().

DDE-Nachricht	Zweck	lParam2
WM_DDE_INITIATE	Fordert den Beginn einer Konversation an.	DDEINIT far *
WM_DDE_INITIATE ACK	Wird von der Server-Anwendung als Antwort auf die WM_DDE_INITIATE-Nachricht für alle Themen gesendet, die der Server unterstützt. Die Anwendung benützt Win-DDEInitiate() zum Senden dieser Nachricht.	DDEINIT far *
WM_DDE_TERMINATE	Wird von irgendeiner Anwendung der DDE-Konversation gesendet um die Konversation zu beenden.	(reserviert)
WM_DDE_REQUEST	Anforderungen der Daten vom Server.	DDESTRUCT far *
WM_DDE_ACK	Verständigt die Anwendung vom Erhalt und der Bearbeitung von: WM_DDE_EXECUTE WM_DDE_DATA WM_DDE_ADVISE WM_DDE_UNADVISE WM_DDE_POKE WM_DDE_REQUEST (in einzelnen Fällen)	DDESTRUCT far *
WM_DDE_DATA	Informiert eine Client-Anwendung über die Verfügbarkeit von Daten.	DDESTRUCT far *
WM_DDE_ADVISE	Fordert die Server-Applikation auf, Daten zu senden und sie zu aktualisieren, wenn sie sich ändern.	DDESTRUCT far *
WM_DDE_UN-ADVISE	Fordert die Server-Anwendung auf, daß die angegebenen Daten nicht mehr aktualisiert werden sollen.	DDESTRUCT far *
WM_DDE_POKE	Fordert eine Anwendung auf, nicht geforderte Daten zu empfangen.	DDESTRUCT far *
WM_DDE_EXECUTE	Sendet einer Server-Anwendung einen String, der als Serie von Kommandos abgearbeitet werden soll.	DDESTRUCT far *





im *Bild 3* sehen. Das System nimmt die Strings, die der Funktion WinDdeInitiate übergeben werden, und kopiert sie in mehrere DDEINIT-Blöcke, die an alle Anwendungen versendet werden, die im System laufen. Die an DDE teilnehmeneden Server-Anwendungen bearbeiten den Funktionsaufruf WinDdeInitiate, indem sie die Nachricht WM_DDE_INITIATEACK mit der Funktion Win-DdeRespond an den Client zurücksenden, und damit das DDEINIT-Speicherobjekt des WinDde-Initiate-Aufrufs freigeben. Die anderen Anwendungen antworten nicht auf die Nachricht WM_DDE_INITIATE, und der gemeinsame Speicher wird von der Standard-Fensterfunktion des Systems freigegeben.

Bild 3 zeigt auch, wie die Struktur DDE-STRUCT von Client- und Server-Anwendung allokiert, übergeben und wieder freigegeben wird. *Bild 4* verdeutlicht die Bearbeitung von DDEINIT.

Mehrere Clients, mehrere Server

Oben haben wir das Ein-Server/Ein-Client-DDE-Konversationsmodell erläutert. Es kann auch möglich sein, und in einer Multitasking-Umgebung wie OS/2 und dem Presentation Manager ist es wahrscheinlich, daß erstens Server-Anwendungen mehrere Client-Anwendungen versorgen müssen und zweitens mehrere Client-Anwendun-

gen Daten gleichzeitig von mehreren Servern erhalten können. Ähnlich gibt es auch nur einen geringen oder gar keinen Unterschied, ob die Client- oder die Server-Anwendung zuerst startet. In einer Umgebung mit mehreren Servern und mehreren Clients können mehrere Server und Client-Anwendungen in jeder denkbaren Reihenfolge mit allen möglichen Permutationen aufgerufen werden. Die wirkliche Stärke von DDE um Konversationen effektiv abzuwickeln, wird erst in einer Implementierung mit mehreren Servern und Clients deutlich.

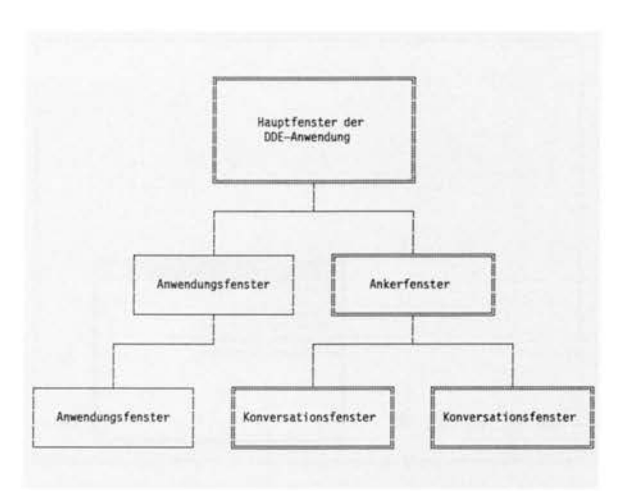
In einer N-Wege-Konversation können Server, Client oder beide Anwendungen in einer 1-zu-N-Konversation arbeiten. Das heißt, ein Server liefert mehreren Client-Anwendungen Daten, ein Client erhält Daten von mehreren Server-Anwendungen, oder beide Zustände treten gleichzeitig auf.

Die DDE-Konvention für die Behandlung einer 1-zu-N-Konversation besteht darin, daß die steuernde Anwendung ein Fenster – das DDE-Konversationsfenster – für jede Konversation öffnet, und die Nachrichten in einer speziellen Fensterprozedur bearbeitet, die auf der Fenster-Handle basiert. Es gibt einige Vorteile, die die Bearbeitung der Konversation auf der Basis der dieser Konversation zugewiesenen Fenster-Handle bietet.

Zuerst dient die individuelle Fenster-Handle als Identifikation für die Konversation. Diese Handle ist eindeutig, was durch das Betriebssystem sichergestellt wird. Zweitens kann die Konversation leicht durch PM-API-Funktionen ausgeführt werden (wie zum Beispiel WinEnumerateWindow), ohne daß spezielle Datenstrukturen wie verkettete Listen für die Überwachung der Konversation entwickelt werden müssen. Drittens ist es leicht möglich, konversationsspezifische Daten in Fenster-Worten, die mit jedem erzeugten Fenster generiert werden, abzulegen und wieder auszulesen. Wir haben in unserer Anwendung ein zusätzliches Fenster in jeder DDE-Anwendung erzeugt, um die DDE-Bearbeitung zu erleichtern.

Jede unserer DDE-Anwendungen erzeugt ein DDE-Ankerfenster, welches eine künstliche einschichtige Fensterhierarchie über der Fensterstruktur der Anwendung darstellt und alle DDE-Fenster unter einem einzigen Elternfenster isoliert. Dies ermöglicht uns die Suche in mehreren DDE-Konversationen durch den Aufruf von WinEnumerateWindow, ohne alle Kinderfenster der Anwendungen durchsuchen zu müssen. Die DDE-Konversationsfenster werden normal durchlaufen, um Informationen zu finden, die zu einer bestimmten Konversation gehören. *Bild 5* zeigt die Eltern-/Kind-Beziehung der DDE-Fenster in unserer Anwendung.

Wenn sowohl der Server, als auch der Client die Konversation initiieren kann, wie dies in einer Multiclient-/Multiserver-Anwendung der



◂ Bild 5: Die Vater-/Sohn-Beziehung der DDE-Fenster unserer Anwendung wird in diesem Stammbaum dargestellt. Um die DDE-Bearbeitung unserer Anwendung zu vereinfachen, werden alle DDE-Fenster unter einem einzelnen Vater-Fenster isoliert.

Fall ist, sollte eine Client-/Server-Beziehung, ähnlich derjenigen, die in der Ein-Server/Ein-Client-DDE-Konversation verwendet wird, gewählt werden.

Wird ein Server während der Ausführung eines Client gestartet, so muß der Server dem Client mitteilen, daß er an der Konversation teilnehmen will. In unserem Fall haben wir die Nachricht WM_DDE_INITIATE mit einem vordefinierten Anwendungsnamen gewählt. Der Client antwortet auf die Initialisierungs-Nachricht des Servers mit einer eigenen Initiierungs-Nachricht, was den Server veranlaßt, mit der Nachricht WinDdeRespond zu antworten, wie dies auch im Ein-Server/Ein-Client-Modell der Fall ist.

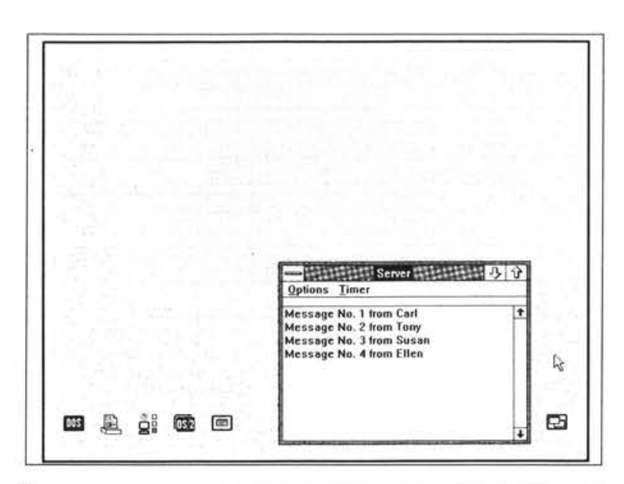
Ein Grafik-austauschprogramm

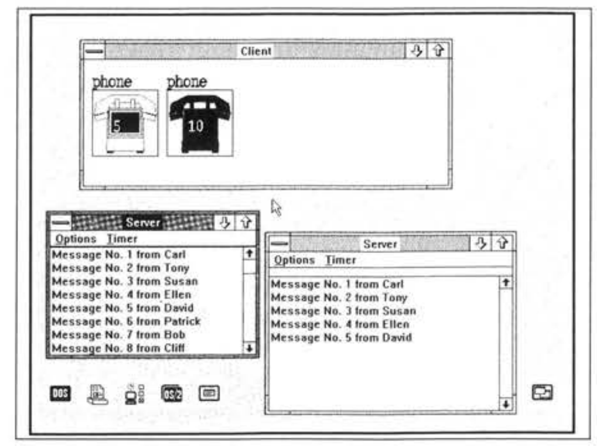
DDE-Erweiterungen für das Multiclient/Multiserver-Modell sind im grafischen Beispiel-Austauschprogramm enthalten. Zur Demonstration zeigt die Client-Anwendung nur grafische Bilder, die die laufende Server-Anwendungen darstellen sollen. Die Server-Anwendungen können sich stark in den Funktionen unterscheiden, die sie zur Verfügung stellen, aber sie benutzen alle DDE, um die grafischen Daten an den Client zu übertragen. In unserem Falle installiert der Client eine permanente Datenverbindung mit dem Server, um Daten zu erhalten, wenn sich der Zustand des Bildes ändert. Wir wählten Graphics_Exchange als Themen-Name für dieses Beispiel.

Der Server in diesem Beispiel simuliert einen Telefon-Nachrichtenservice. Bei der Ankunft von Nachrichten werden diese im Hauptfenster der Server-Anwendung angezeigt. Bei jeder neuen Nachricht wird das Bild erneuert, so daß es die augenblickliche Anzahl der Telefonnachrichten widerspiegelt. Die Client-Anwendung erhält eine WM_DDE_DATA-Nachricht. Um die Komplexität der Server-Anwendung einzuschränken, werden die Nachrichten durch den Timer ausgelöst. In regelmäßigen Intervallen werden Telefonnachrichten in die Liste eingefügt oder aus ihr ge-

▸ *Bild 6: Aufruf des Servers.*

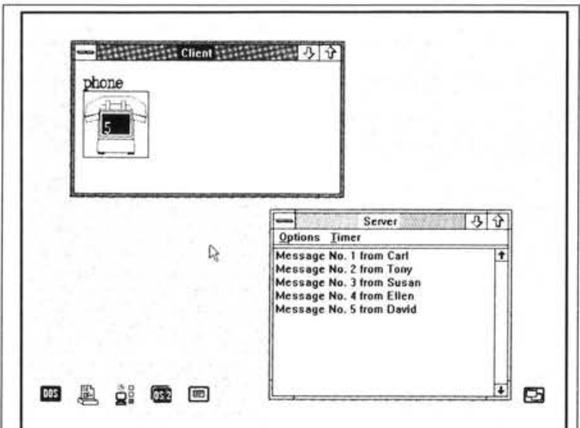
▸▸ *Bild 8: Der zweite Server ist gestartet und signalisiert dem Client.*





▸ *Bild 7: Aufruf des Client und DDE-Initialisierung.*

▸▸ *Listing 9: Initialisierung der DDE-Konversation Graphics_Exchange.*




```
case WM_CREATE:  /* broadcast the initiate message  */
     WinDdeInitiate(hwnd, "", "Graphics_Exchange");
     break;

case WM_DDE_INITIATE: /* reply with an initiate to specific server */
   if (hwnd != lParam1) {
       if ((!strcmp("Client", DDEInitPtr->pszAppName)) &&
           (!strcmp("Graphics_Exchange",DDEInitPtr->pszTopic))) {
           itoa((int)lParam1, itoa_buf, 10);
           WinDdeInitiate(hwnd, itoa_buf, "Graphics_Exchange");
       }
   }
   DosFreeSeg(PDDEITOSEL(DDEInitPtr));
   break;
```

löscht. Dadurch können wir uns auf die DDE-Implementierung konzentrieren, statt auf die Funktionen der Server-Anwendung.

In *Bild 6* ist die Server-Anwendung gestartet und erzeugt Telefonnachrichten. Beim Aufruf der Client-Anwendung versendet diese die Nachricht WM_DDE_INITIATE, und der Server antwortet. Beachten Sie, daß der Anwendungsstring ein Nullstring ist, um anzugeben, daß der Client mit jeder Anwendung kommunizieren möchte, die auf das Thema Graphics_Exchange antwortet. *Bild 7* zeigt den Bildschirm, nachdem die Konversation aufgebaut ist und das erste Bild an den Client übertragen wurde.

Nach dem Aufruf einer zweiten Server-Anwendung muß der Server dem Client mitteilen, daß er die Ausführung gestartet hat. In unserem Fall versendet der Server die Nachricht WM_DDE_INITIATE mit denselben Themen, aber mit der Angabe einer Zielanwendung des Client. Dies zeigt, daß die Initiierung ein spezieller Fall ist, bei der der Server seinen Aufruf jeder Client-Anwendung anzeigt, die eventuell später eine Konversation aufbauen möchte. Nach Erhalt der Nachricht versendet die Client-Anwendung eine weitere WM_DDE_INITIATE-Nachricht. Wäre aber jetzt der String mit dem Anwendungsnamen ein Nullstring, so würde unbeabsichtigt mit der Original-Server-Anwendung eine zweite Verbindung aufgebaut. Um diesen Fall auszuschließen, gibt der Client einen Anwendungsnamen an, der einfach die in einen String gewandelte Handle des neuen Servers ist. Der neu gestartete Server prüft den Anwendungsnamen und antwortet, da der Anwendungsname mit seiner Handle übereinstimmt. *Bild 8* zeigt den Datenfluß beim Aufbau der zweiten Verbindung.

Diese Signalisierungskonvention führt uns zu einigen Regeln für die Initiierung der Graphics_Exchange-Konversation und die folgende Antwort. Die Client-Anwendung muß beim Start eine WM_DDE_INITIATE-Nachricht versenden, die als Anwendungsnamen einen Nullstring enthält. Sie muß aber auch eine Nachricht WM_DDE_INITIATE von einer neu aufgerufenen Anwendung empfangen können. Ist der Anwendungsname Client in dieser Nachricht enthalten, dann muß der Client wieder eine WM_DDE_INITIATE-Nachricht senden, aber diesmal mit einem Anwendungsstring, der die Handle des neuen Teilnehmers enthält. *Listing 9* zeigt die Anfangsbearbeitung der Client-Anwendung Graphics_Exchange.

Der Server darf auf die WM_DDE_INITIATE-Nachricht mit dem Thema Graphics_Exchange nur dann antworten, wenn die Stringlänge des Anwendungsnamens 0 ist, oder die Stringdarstellung der Fenster-Handle des Servers enthält. Alle anderen Anwendungsnamen sollten ignoriert werden. Antwortet der Server auf die Konversation, so erzeugt er ein Fenster, um die weitere Bearbeitung der neuen Konversation handhaben zu können, und antwortet dem Client mit dieser Fenster-Handle. Beachten Sie, daß ein Fensterwort, das den Verbindungszähler enthält, inkrementiert wird. Dieser Zähler wird später vom Server benötigt, um zu entscheiden, wann der Abbruch erfolgen darf. *Listing 10* zeigt die Antwortbearbeitung der Server-Anwendung Graphics_Exchange.

```
case WM_DDE_INITIATE:  /* respond if app and topic strings match */
     pDDEInit = (PDDEINIT)lParam2;

   /* check for null strings or Graphics_Exchange */
   if ((HWND)lParam1 != hwnd) {
       itoa((int)hwnd, szTemp, 10);
       if(((!strlen(pDDEInit->pszAppName)) &&
           (!strcmp("Graphics_Exchange", pDDEInit->pszTopic))) ||
           ((!strcmp(szTemp, pDDEInit->pszAppName)) &&
           (!strcmp("Graphics_Exchange", pDDEInit->pszTopic))) ||
           ((!strlen(pDDEInit->pszTopic)) &&
           (!strlen(pDDEInit->pszAppName))))    {

           /* create conversation window and respond */
           hwndConv = WinCreateWindow(hwndDDE,
                                      (PSZ)"DDEConversation",
                                      (PSZ)NULL, WS_VISIBLE,
                                      0,0,0,0, hwnd, HWND_TOP,
                                      ++nConvID, (PVOID)NULL,
                                      (PVOID)NULL);
           pWWVar->ConvCnt++;
           WinDdeRespond(lParam1, hwndConv, "Client",
                         "Graphics_Exchange");
      }
   }
   DosFreeSeg(PDDEITOSEL(pDDEInit));
   break;
```

◄◄ *Listing 10: Antwort auf die DDE-Konversation Graphics_Exchange.*

`BOOL WinDdeInitiate(hwndClient, pszAppName, pszTopicName)`
Schickt die Nachricht WM_DDE_INITIATE an das Hauptfenster aller Anwendungen. Diese Funktion erzeugt eine DDEINIT-Struktur und übergibt den Zeiger dieser Struktur als lParam2 der Initialisierungsnachricht.

`MRESULT WinDdeRespond(hwndClient, hwndServer, pszAppName, pszTopicName)`
Sendet die Nachricht WM_DDE_INITIATEACK zurück an den Sender der WM_DDE_INITIATE-Nachricht. Die Funktion erzeugt eine DDEINIT-Struktur und übergibt einen Zeiger auf die Struktur in lParam2 der Initialisierungs-Bestätigungsnachricht.

`BOOL WinDdePostMsg(hwndTo, hwndFrom, ULONG wm, DDESTRUCT far *, BOOL fRetry)`
Sendet eine Nachricht an hwndTo; wm enthält die Nachricht, die im Bereich WM_DDE_FIRST und WM_DDE_LAST liegen muß. Wenn fRetry FALSE ist, so gibt die Funktion FALSE zurück, wenn die Nachricht nicht richtig gesendet werden konnte. Ist fRetry TRUE wird das Senden der Nachricht im Abstand von einer Sekunde wiederholt, bis sie richtig gesendet wurde.

◄◄ *Tabelle 2: Die DDE-Funktionen des Presentation Managers.*

Bei Erhalt der Nachricht WM_DDE_INITIATE-ACK erzeugt die Client-Anwendung ein Fenster, um die Verbindung zu bearbeiten. Ebenso wie der Server, erhöht der Client einen Konversations-Verbindungszähler, der in seinem Fensterwort enthalten ist. Dies wird bei der Programmende-Bearbeitung benötigt. Die Nachrichten WM_DDE_REQUEST und WM_DDE_ADVISE werden an den Server auf Anforderung des neu erzeugten Konversationsfensters gesendet. *Listing 11* zeigt diese Bearbeitung.

DDESTRUCT wird für alle Meldungen bis auf WM_DDE_INITIATE und WM_DDE_INITIATE-ACK verwendet.

```
typedef struct _DDESTRUCT {
        ULONG    cbData;        /* allocated size of data     */
        USHORT   fsStatus;      /* status flags               */
        USHORT   usFormat;      /* DDE format                 */
        USHORT   offszItemName; /* offset to item referred to
                                   in this message            */
        USHORT   offabData;     /* offset to beginning of data */
} DDESTRUCT;
```

Der Item-Stringname und die tatsächlichen Daten werden an DDESTRUCT im selben Speicherobjekt angehängt.

```
BYTE    szItemName[];    /* null terminated item name string*/
BYTE    abData[];        /* actual data */
```

DDEINIT wird bei den Meldungen WM_DDE_INITIATE und WM_DDE_INITIATE-ACK gesendet.

```
typedef struct _DDEINIT {
        USHORT        cb;            /* size of memory object */
        PSZ           pszAppName;    /* pointer to application
                                        name string in memory object */
        PSZ           pszTopic;      /* pointer to topic name
                                        string in memory object */
} DDEINIT;
```

◄ *Tabelle 3: Die DDE-Datenstrukturen des Presentation Managers.*

```
case WM_DDE_INITIATEACK:  /* establish conversation with server */
     if( ((!strcmp("Client", DDEInitPtr->pszAppName)) &&
         (!strcmp("Graphics_Exchange", DDEInitPtr->pszTopic))) ||
        ((!strlen(DDEInitPtr->pszAppName)) &&
         (!strcmp("Graphics_Exchange",DDEInitPtr->pszTopic)))) {
        /* create a window for the conversation  -
           child of DDEanchorHWND  */
        DDEconversationHWND = WinCreateWindow(hwnd, (PSZ)"DDE_Win",
                                 (PSZ)NULL, WS_VISIBLE, 0, 0, 0, 0,
                                 WinQueryWindow(hwnd,QW_PARENT,FALSE),
                                 HWND_TOP, ++winDDEid, (PVOID)NULL,
                                 (PVOID)NULL);
        WinSetWindowULong(DDEconversationHWND, WW_CONV_HWND,
                          (ULONG)lParam1);
        WinSetWindowULong(WinQueryWindow(hwnd, QW_PARENT, FALSE),
                          WW_CONVCOUNT,
                          WinQueryWindowULong(
                              WinQueryWindow(
                                  hwnd,QW_PARENT,
                                  FALSE),
                          WW_CONVCOUNT)+1);
        /* send a request for initial data */
        DDEstrptr = st_DDE_Alloc(sizeof(DDESTRUCT) +
                    strlen("Graphics")+1,"DDEFMT_graphics_data");
        DDEstrptr->offszItemName = (USHORT)sizeof(DDESTRUCT);
        strcpy(DDES_PSZITEMNAME(DDEstrptr), "Graphics");
        WinDdePostMsg((HWND)lParam1, DDEconversationHWND,
                    (ULONG)WM_DDE_REQUEST, DDEstrptr, TRUE);
        /* send an advise to subscribe to
           receive future data updates */
        DDEstrptr = st_DDE_Alloc(sizeof(DDESTRUCT) +
                    strlen("Graphics")+1,"DDEFMT_graphics_data");
        DDEstrptr->offszItemName = (USHORT)sizeof(DDESTRUCT);
        strcpy(DDES_PSZITEMNAME(DDEstrptr), "Graphics");
        WinDdePostMsg((HWND)lParam1, DDEconversationHWND,
                    (ULONG)WM_DDE_ADVISE, DDEstrptr, TRUE);
     }     /* free the memory */
     DosFreeSeg(PDDEITOSEL(DDEInitPtr));
     break;
```

◄ *Listing 11: Herstellen einer Verbindung in der DDE-Konversation Graphics_Exchange.*

Die primäre Aktivität des Server-Fensters besteht im Packen der Daten und Senden der WM_DDE_DATA-Nachricht. Im Beispiel wird diese Aktivität immer bei der Bearbeitung von WM_DDE_REQUEST ausgeführt. Das heißt, sogar wenn die Hauptserver-Anwendung entscheidet, daß wegen einer vorhergehenden ADVISE-Nachricht Daten gesendet werden sollen, wird

▸ *Listing 12: Bearbeitung von WM_DDE_REQUEST im Graphics_Exchange-Server.*

```
case WM_DDE_REQUEST: /* allocate DDESTRUCT and dump graphics data */
     pDDEStruct = (PDDESTRUCT)lParam2;
     strcpy(szTemp, "Graphics");
     if((pDDEStruct->usFormat == pWWVar->usFormat) &&
          (!strcmp(szTemp, DDES_PSZITEMNAME(pDDEStruct)))) {
           if(!pWWVar->bNoData) {
              nNumBytes = (strlen(szTemp) + 1 + sizeof(GDEDATA) +
                           LOUSHORT(lPhFigCnt));
              pDDEStruct = st_DDE_Alloc((sizeof(DDESTRUCT) +
                            nNumBytes), "DDEFMT_graphics_data");
              pDDEStruct->cbData = sizeof(GDEDATA) + lPhFigCnt;
              pDDEStruct->offszItemName = (USHORT)sizeof(DDESTRUCT);
              pDDEStruct->offabData = (USHORT)((sizeof(DDESTRUCT) +
                            strlen(szTemp)) + 1);
              pGDEData = (PGDEDATA)DDES_PABDATA(pDDEStruct);
              st_Init_GDEData(pGDEData);
              pGDEData->cBytes = lPhFigCnt;
              strcpy(pGDEData->szItem, "phone");
              pGDEData->pGpi = (unsigned char far *)((LONG)pGDEData +
                                sizeof(GDEDATA));
              GpiGetData(hpsGraphics, (LONG)IDSEG_PHONE,
                         (PLONG)&lOffset, DFORM_NOCONV,
                         (LONG)lPhFigCnt, (PBYTE)pGDEData->pGpi);
              memcpy(DDES_PSZITEMNAME(pDDEStruct), szTemp,
                     (strlen(szTemp) + 1));
              if(lParam1 != WinQueryWindow(hwnd, QW_OWNER, FALSE)) {
                 pDDEStruct->fsStatus |= DDE_FRESPONSE;
                 pWWVar->hwndLink = (HWND)lParam1;
              }
              WinDdePostMsg(pWWVar->hwndLink, hwnd, WM_DDE_DATA,
                            pDDEStruct, TRUE);
          }
          else {
              WinDdePostMsg(pWWVar->hwndLink, hwnd, WM_DDE_DATA,
                            NULL, TRUE);
          }
          DosFreeSeg(PDDESTOSEL(lParam2));
     }
     else {  /* post negative ACK using their DDESTRUCT */
          pDDEStruct->fsStatus &= (~DDE_FACK);
          WinDdePostMsg(lParam1, hwnd, WM_DDE_ACK, pDDEStruct, TRUE);
     }
     break;
```

▸ *Listing 13: Erzeugen mehrerer WM_DDE_REQUEST-Nachrichten während ADVISE.*

```
case WM_TIMER:  /* insert phone message into listbox,
                   update picture, and generate new
                   REQUESTS for all ADVISING windows */
     .
     .  /* listbox and picture have been updated */
     .
     hEnum = WinBeginEnumWindows(hwndDDE);
     while((hwndEnum = WinGetNextWindow(hEnum))) {
        pWWChild = (PWWVARS)WinQueryWindowULong(hwndEnum, QWL_USER);
        if (pWWChild->bAdvise) {
            pDDEStruct = st_DDE_Alloc(sizeof(DDESTRUCT) +
                          strlen("Graphics")+1,
                          "DDEFMT_graphics_data");
            pDDEStruct->offszItemName = (USHORT)sizeof(DDESTRUCT);
            strcpy(DDES_PSZITEMNAME(pDDEStruct), "Graphics");
            WinDdePostMsg(hwndEnum, hwnd, WM_DDE_REQUEST,
                          pDDEStruct, TRUE);
        }
        WinLockWindow(hwndEnum, FALSE);
     }
     WinEndEnumWindows(hEnum);
     break;
 .
```

▸▸ *Listing 14: Die Bearbeitung von WM_DDE_ADVISE und WM_DDE_UNADVISE.*

```
case WM_DDE_ADVISE:  /* set ADVISE bit in window data and ACK */
     pDDEStruct = (PDDESTRUCT)lParam2;
     if(pDDEStruct->usFormat == pWWVar->usFormat) {
        pWWVar->hwndLink = (HWND)lParam1;
        pWWVar->bAdvise = TRUE;
        if(pDDEStruct->fsStatus & DDE_FNODATA) {
          pWWVar->bNoData = TRUE;
        }
        if(pDDEStruct->fsStatus & DDE_FACKREQ) {
           pDDEStruct->fsStatus |= DDE_FACK;
           WinDdePostMsg(pWWVar->hwndLink, hwnd, WM_DDE_ACK,
                         pDDEStruct, TRUE);
        }
        else {
           DosFreeSeg(PDDESTOSEL(lParam2));
        }
     }
     else {    /* Send a negative ACK using their DDEStruct */
        pDDEStruct->fsStatus &= (~DDE_FACK);
        WinDdePostMsg(lParam1, hwnd, WM_DDE_ACK, pDDEStruct, TRUE);
     }
     break;

case WM_DDE_UNADVISE:/* turn off ADVISE bit in window data and ACK */
   pDDEStruct = (PDDESTRUCT)lParam2;
   if((lParam1 == pWWVar->hwndLink) &&
      (pDDEStruct->usFormat == pWWVar->usFormat)) {
       pWWVar->bAdvise      = FALSE;
       pWWVar->bNoData      = TRUE;
       pDDEStruct->fsStatus |= DDE_FACK;
       WinDdePostMsg(lParam1, hwnd, WM_DDE_ACK, pDDEStruct, TRUE);
     }
   else {
       pDDEStruct->fsStatus &= (~DDE_FACK);
       WinDdePostMsg(lParam1, hwnd, WM_DDE_ACK, pDDEStruct, TRUE);
   }
   break;
```

eine WM_DDE_REQUEST-Nachricht von der Hauptanwendung an alle Konversationsfenster gesendet, die unterrichtet werden. Das Format der Datenübertragung ist ein benutzerdefiniertes Format, das wie früher besprochen registriert wird.

Das Datenformat ist in Wirklichkeit die Datenstruktur GDEDATA, die als Eingabestruktur benützt wird, die die Grafik für die Client-Anwendung manipuliert. Nach Erhalt der Nachricht WM_DDE_REQUEST allokiert der Server Speicher für DDESTRUCT und die darunterliegende Datenstruktur und trägt die notwendigen Daten, die vom Client gebraucht werden, ein. Die eigentlichen grafischen Daten können als Bitmap oder GPI-Zeichenanweisungen vorhanden sein, und sind im Speicherobjekt enthalten. Im Beispiel werden sie immer als Zeichenanweisungen abgelegt. Das ganze Datenpaket wird dann durch den Funktionsaufruf WinDdePostMsg übertragen. Die komplette Bearbeitung der Nachricht WM_DDE_REQUEST ist im *Listing 12* dargestellt.

Listing 13 zeigt, wie die Hauptserver-Anwendung die Nachricht WM_DDE_REQUEST für die Client-Anwendungen erzeugt, wenn sich die Daten ändern. Dies wird durch Aufzählen aller Kinderfenster des Hauptfensters und Senden an die Fenster, die zu einer Client-Anwendung gehören, erreicht. Der Informationsstatus der Server-Anwendung wird im Fensterwort der Konversations-Fensterprozedur gespeichert. *Listing 14* zeigt das Setzen des Statusfeldes nach Erhalt der Nachricht WM_DDE_ADVISE oder WM_DDE_-UNADVISE.

Das Datenformat für eine Graphics_Exchange-Konversation ist einfach die Eingabestruktur einer Steuerung. Daher besteht die WM_DDE_DATA-Bearbeitung bei einer Client-Anwendung nur aus einem Einsetzen oder Ersetzen der Datenstruktur in eine Steuerung. Ist das DDE_RESPONSE-Bit gesetzt, weiß die Client-Anwendung, daß die Nachricht WM_DDE_DATA eine Folge der ursprünglichen Nachricht WM_-DDE_REQUEST ist, die nach dem Verbindungsaufbau gesendet wurde. Da dies die erste Übertragung von Daten vom Server ist, muß das Bild in die grafische Steuerung eingefügt werden. Ist das DDE_RESPONSE-Bit nicht gesetzt, so ist die Nachricht WM_DDE_DATA eine Folge einer ADVISE-Nachricht, und der Client muß das Bild durch neue Daten ersetzen.

Die Kontrollnachrichten selbst sind sehr einfach. Der erste Parameter ist die Identifikation des fraglichen Bildes, und der zweite zeigt auf eine Struktur, die das Bild beschreibt und ent-

```
DDEstrptr = (PDDESTRUCT)lParam2;

switch (message){

   case WM_DDE_DATA:   /* process incoming picture */
       gde_ptr = (PGDEDATA)DDES_PABDATA(DDEstrptr);
       gde_ptr->hwnd_idItem = LOUSHORT(hwnd);

       /*  add if request  * /
       if (DDEstrptr->fsStatus && DDE_FRESPONSE) {
           WinSendMsg(WinWindowFromID(
                       WinQueryWindow(hwnd, QW_OWNER, FALSE),
                       ID_GRAPHICS1),IC_INSERTITEM,
                       MPFROMSHORT(ICM_END), (MPARAM)gde_ptr);
       } else {  /* replace if advise */
           WinSendMsg(WinWindowFromID(
                       WinQueryWindow(hwnd,QW_OWNER,FALSE),
                       ID_GRAPHICS1),IC_SETITEMSTRUCT,
                       MPFROMSHORT(gde_ptr->hwnd_idItem),
                       (MPARAM)gde_ptr);
       }

       if (DDEstrptr->fsStatus && DDE_FACKREQ) {
           DDEstrPtrAck = st_DDE_Alloc(sizeof(DDESTRUCT),
                            "DDEFMT_graphics_data");
           WinDdePostMsg((HWND)lParaml, hwnd, (ULONG)WM_DDE_ACK,
                            DDEstrPtrAck, TRUE);
       }

       DosFreeSeg(PDDESTOSEL(DDEstrptr));

       break;
```

◂◂ *Listing 15: WM_DDE_DATA-Bearbeitung des Graphics_Exchange-Client.*

Name des Flags	Zweck
DDE_FACK	Gesetzt bei positivem ACK.
DDE_FBUSY	Gesetzt, falls die Anwendung beschäftigt ist.
DDE_FNODATA	Die Anwendung hat keine Daten für ADVISE.
DDE_FACKREQ	Gesetzt, falls die Anwendung ACKs verlangt.
DDE_FRESPONSE	Gesetzt, falls die Nachricht eine Antwort auf REQUEST ist.
DDE_NOT-PROCESSED	Nachricht wird von der Anwendung nicht unterstützt.
DDE_FRESERVED	Reserviert.
DDE_APPSTATUS	Anwendungsspezifische Rückgabe.

◂◂ *Tabelle 4: Die Flags im Feld fsStatus der Struktur DDESTRUCT.*

Makroname	Zweck
DDES_PSZITEM-NAME(pddes)	Gibt szItemName aus der DDESTRUCT pddes zurück.
DDES_PAB-DATA(pddes)	Gibt einen Far-Zeiger auf den Datenbereich zurück, der der DDESTRUCT pddes folgt.
SELTOPDDES(sel)	Konvertiert einen Selektor in einen Far-Zeiger.
PDDESTO-SEL(pddes)	Konvertiert einen Far-Zeiger auf DDESTRUCT pddes in einen Selektor. Dies wird beim Gebrauch von DosFreeSeg benötigt, um eine Struktur freizugeben.
PDDEITO-SEL(pddei)	Konvertiert einen Far-Zeiger auf DDEINIT pddei in einen Selektor. Dies wird beim Gebrauch von DosFreeSeg benötigt, um eine Struktur freizugeben.

◂ *Tabelle 5: DDE-Makros für die Datenstrukturen DDEINIT und DDESTRUCT.*

```
/* send WM_DDE_UNADVISE, shutdown all conversations
   for this application, then quit */
case WM_CLOSE:
   if (WinQueryWindowULong(hwnd, WW_CONVCOUNT)) {
       WinSetWindowULong(hwnd, WW_CLOSE,
                         WinQueryWindowULong(
                          hwnd, WW_CLOSE) | WIN_CLOSING_FLAG);
       henum = WinBeginEnumWindows(DDEanchorHWND);
       while (hwndenum = WinGetNextWindow(henum)) {
           WinSetWindowULong(hwndenum, WW_CONV_FLAGS,
                     WinQueryWindowULong(hwndenum, WW_CONV_FLAGS) |
                     WIN_TERM_FLAG);
           tohwnd = (HWND)WinQueryWindowULong(hwndenum,WW_CONV_HWND);
           DDEptr = st_DDE_Alloc(sizeof(DDESTRUCT) +
                       strlen("Graphics")+1,"DDEFMT_graphics_data");
           DDEptr->offszItemName = (USHORT)sizeof(DDESTRUCT);
           strcpy(DDES_PSZITEMNAME(DDEptr), "Graphics");
           WinDdePostMsg(tohwnd, hwndenum, WM_DDE_UNADVISE,
                         DDEptr, TRUE);
           WinDdePostMsg(tohwnd, hwndenum, WM_DDE_TERMINATE,
                         NULL, TRUE);
           WinLockWindow(hwndenum, FALSE);
       }
       WinEndEnumWindows(henum);
   }
   else {
       WinPostMsg(hwnd,WM_QUIT,0L,0L); /* quit if no
                                          conversations open  */
   }
   break;
```

◂ *Listing 16: WM_CLOSE-Bearbeitung und Versenden der Nachricht WM_DDE_TERMINATE.*

hält. Die internen Details der Steuerung sind nicht wichtig. Die Steuerung handhabt die Plazierung der grafischen Daten. Die komplette WM_DDE_DATA-Bearbeitung sehen Sie im *Listing 15*.

Während der Ausführung der beteiligten Anwendungen wird die Nachricht WM_DDE_DATA jedesmal gesendet, wenn beim Timer-Tick das Einfügen oder Löschen einer Telefon-Nachricht festgestellt wird. Die DDE-Konversation dauert an, bis eine der Anwendungen sie beendet. In unserem Fall passiert das nur, wenn der Benutzer eine Anwendung abbricht. Die Bearbeitung von WM_CLOSE und die folgende von WM_DDE_TERMINATE sind im Server und im Client gleich.

Wird eine WM_CLOSE-Nachricht erhalten, zählt die Anwendung alle DDE-Konversationsfenster durch. In jedem Konversationsfenster wird ein Fensterwort gesetzt, um anzuzeigen, daß der Benutzer die Anwendung beenden möchte. Zur selben Zeit wird ein anderes Fensterwort nach der Handle des Konversationsfensters, mit dem das Fenster Daten austauscht, abgefragt. Dem Fenster wird eine WM_DDE_TERMINATE-Nachricht von der Hauptanwendung über das Konversationsfenster gesendet. Keine weitere Bearbeitung wird ausgeführt, bis nicht alle Konversationsverbindungen beendet sind. *Listing 16* zeigt die Bearbeitung von WM_CLOSE durch die Client-Anwendung. Beachten Sie, daß der Programmteil des Servers sehr ähnlich ist.

Beim Erhalt der Nachricht WM_DDE_TERMINATE prüft ein Konversationsfenster sein Fensterwort, um festzustellen, ob das Ende von der eigenen Anwendung oder vom Konversationspartner veranlaßt wurde. Wurde es vom Konversationspartner veranlaßt, sendet das Fenster eine zugehörige WM_DDE_TERMINATE-Nachricht an den Sender. Wurde das Ende nicht vom Konversationspartner ausgelöst, ist die Nachricht nur eine Bestätigung des Partners, daß die Programmende-Bearbeitung weitergehen kann. In jedem Fall kann sich das Konversationsfenster jetzt zerstören, da die Konversationsverbindung beendet ist. Zu diesem Zeitpunkt sendet es eine anwendungsdefinierte Meldung, APPM_CONV_CLOSE, um dem Hauptfenster anzuzeigen, daß die Verbindung ordentlich beendet ist. *Listing 17* zeigt die Bearbeitung von WM_DDE_TERMINATE durch die Client-Anwendung. Auch dieser Programmteil ist im Server gleich.

► *Listing 17: WM_DDE_TERMINATE-Bearbeitung und Versenden der Nachricht APPM_CONV_CLOSE.*

```
/* post terminate to server, tell client, and die */
case WM_DDE_TERMINATE:
   if (!WinQueryWindowULong(WinQueryWindow(hwnd, QW_OWNER, FALSE),
                            WW_CLOSE)) {
      WinDdePostMsg((HWND)lParam1, hwnd, WM_DDE_TERMINATE,
                    NULL, TRUE);
   }
   WinPostMsg(WinQueryWindow(hwnd,QW_OWNER,FALSE),
              APPM_CONV_CLOSE,MPFROMLONG(hwnd),(MPARAM)NULL);
   WinDestroyWindow(hwnd);

   break;
```

► *Listing 18: Bearbeitung von APPM_CONV_CLOSE und Programmende.*

```
/* decrement conversation count and delete picture */
case APPM_CONV_CLOSE:
   WinSetWindowULong(hwnd,WW_CONVCOUNT,
                     WinQueryWindowULong(hwnd, WW_CONVCOUNT) - 1);
   WinSendMsg(WinWindowFromID(hwnd,ID_GRAPHICS1),IC_DELETEITEM,
                              (MPARAM)LOUSHORT(lParam1),
                              (MPARAM)NULL);
   if (WinQueryWindowULong(hwnd,WW_CLOSE) &&
      !WinQueryWindowULong(hwnd, WW_CONVCOUNT)){
         WinPostMsg(hwnd,WM_QUIT,0L,0L);
   }
   break;
```

Die Nachricht APPM_CONV_CLOSE dient als Signal, daß die abschließende Bearbeitung beginnen kann. Bei jedem Erhalt einer APPM_CONV_CLOSE-Nachricht wird der Verbindungszähler vermindert. Ist er 0, so prüft die Hauptanwendung ihr Fensterwort, um zu sehen, ob das Ende der Anwendung gewünscht wird. Ist dies der Fall, und alle Verbindungen sind erfolgreich abgebaut, so kann die Anwendung ihre Endebearbeitung erledigen und sich dann selbst beenden (*Listing 18*).

Graphics_Exchange bietet ein sehr anschauliches Beispiel für den Gebrauch von DDE, um ständige Verbindungen zwischen einzelnen PM-Anwendungen herzustellen. Unterstützung für mehrere Konversationen ist durch Definieren einer zusätzlichen Nachricht sowie Gebrauch des Fensterworts und der Aufzählung der Fenster gegeben. Natürlich können diese Anwendungen auch auf die zusätzliche Übertragung anderer als den grafischen Daten erweitert werden.

Wenn Sie dieses Beispiel als Anleitung nehmen, so sollten Sie eine generelle Implementierung für Ihren eigenen Multitasking-Datenaustausch zwischen unterschiedlichen Programmen vornehmen können. Wir halten DDE für eine mächtige und flexible Errungenschaft des Presentation Managers, die eine neue Dimension für die Entwicklung von zusammenwirkenden Anwendungen in der Multitaskingumgebung von OS/2 eröffnet. Da die Programmierumgebung sich von Windows zu den vielfältigen Möglichkeiten des OS/2 Presentation Managers gewandelt hat, hat sich auch das DDE-Protokoll gewandelt. Die Änderungen bieten einen neuen Rahmen für zukünftige Erweiterungen, wenn sich die Umgebung weiterentwickelt.

Susan Franklin, Tony Peters

►► *Tabelle 6: Grafische Kontrollnachrichten.*

Nachrichten werden an die Steuerung mit WinSendMsg gesendet, bei der der Nachrichtenparameter der jeweilige Nachrichtencode ist, wie zum Beispiel IC_INSERTITEM. lParam1 und lParam2 werden für die Nachrichten folgendermaßen initialisiert:

Nachricht und Parameter	**Ursache und Verarbeitung**
IC_INSERTITEM	Fügt ein Objekt in die grafische Steuerung ein.
lparam1	Diese Nachricht fügt ein Objekt in die grafische Steuerung ein. iPosition ist die bei 0 beginnende Position, wo das Objekt eingefügt werden soll.
(SHORT)iPosition	Wenn iPosition ICM_END ist, dann wird das Objekt an das Ende angefügt.
lparam2	pItemStruct ist ein Zeiger auf die Struktur GDEDATA. iItemActual ist die Position, an der das Objekt eingefügt wurde.
(PGDEDATA)pItem Struct	Ist die Allokierung für den Speicher, der zum Einfügen benötigt wird, nicht erfolgreich, so wird ICM_MEMERROR zurückgegeben.
Gibt (SHORT)iItem-Actual zurück.	Ist der gewählte Index ungültig, so wird ICM_INVINDEX zurückgegeben.
IC_DELETEITEM	Löscht einen Eintrag aus der grafischen Steuerung.
lparam1	Diese Nachricht löscht einen Eintrag aus der grafischen Kontrolle.
(SHORT)idItem	IdItem ist die eindeutige Identifizierung für den zu löschenden Eintrag.
lparam2 NULL (reserviert) Gibt (SHORT) cItemsLeft zurück	cItemsLeft ist die Anzahl der Einträge, die nach dem Löschen in der Liste verbleiben.
IC_SETITEM-STRUCT	Setzt den Zeiger der Struktur, die durch idItem identifiziert wird auf die Struktur pItemStruct.
lparam1	Gibt TRUE zurück, falls erfolgreich, ansonsten FALSE.
(USHORT) idItem lparam2 (PGDEDATA) pItemStruct Gibt (BOOL) bSuccess zurück	

Entwurf und Erstellung von Programmen mit mehreren Threads mit Microsoft C:

Multithread-Programme unter OS/2

Aus der Sicht eines Programmierers gleicht OS/2 einer neuen Programmiersprache. Um mit dem Betriebssystem vertraut zu werden, müssen Sie Programme schreiben. In diesem Artikel besprechen wir die Installation und Möglichkeiten des Microsoft C-Compilers Version 5.1 für die OS/2-Entwicklung.

◄ *Das Multithread-OS/2-Programm Hello.*

Außerdem betrachten wir die vorhandenen Headerdateien näher. Weiter sehen wir uns die Programmierschnittstelle für Applikationen (API) an, indem wir unser erstes OS/2-Programm schreiben. Am Schluß erkunden wir die Programmierung von Applikationen mit mehreren Threads und schreiben eine Multithread-Version des beliebten HELLO.C-Programms.

Installation des Compilers

Vor der Installation sollten Sie sich vergewissern, daß Sie genügend freien Speicherplatz auf der Platte zur Verfügung haben. Die Mindestgröße für die Version im geschützten Modus ist ungefähr 3,5 Mbyte (mit zwei Bibliotheken). Wollen Sie den gesamten Compiler im geschützten Modus installieren, sollten Sie eher 4,5 Mbyte frei haben. Planen Sie, den Compiler unter DOS und OS/2 zu benutzen, so werden über 6 Mbyte benötigt. Dem Installationsprogramm kann mitgeteilt werden, daß es nur die Teile installiert, die Sie benötigen.

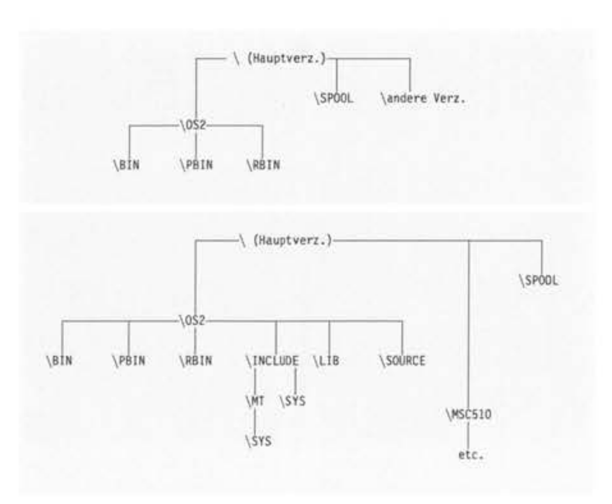
Sie müssen Sich auch für die Verzeichnisstruktur des Compilers entscheiden. Wir erörterten die von OS/2 bevorzugte Verzeichnisstruktur, die im *Bild 1* gezeigt ist, im ersten Artikel dieser Serie (MSJ Mai/Juni '89, Seite 28). Sie werden sich erinnern, daß das Verzeichnis OS2\PBIN die nur im geschützten Modus lauffähigen Programme enthält. OS2\RBIN enthält die Programme für den nicht geschützten Modus und OS2\BIN enthält kombinierte Programme für DOS und OS/2.

Wenn Sie den Compiler unter OS/2 installieren, sehen Sie, daß das Installationsprogramm die Vorschläge der Verzeichnisstruktur in der Weise macht, wie sie in *Bild 2* gezeigt wird. Das Verzeichnis \OS2\LIB enthält alle Bibliotheken des Compilers, \OS2\INCLUDE enthält die Headerdateien und OS2\SOURCE enthält die zusätzliche Dokumentation und die Quelldateien. Beachten Sie auch, daß der Compiler eine zweite Version der Headerdateien unter dem Verzeichnis \OS2\INCLUDE\MT installiert. Dies sind die Headerdateien der Bibliotheken für Programme mit mehreren Threads, die wir uns später ansehen.

▶ *Bild 1: Die OS/2-Verzeichnis-Struktur vor der Installation des Compilers.*

▶ *Bild 2: Die OS/2-Verzeichnisstruktur nach der Installation des Compilers zeigt eine mögliche Verzeichnisorganisation. Die MAKE-Dateien für diesen Artikel basieren auf dieser Struktur. Eine Alternative wäre das Installieren der Unterverzeichnisse INCLUDE, LIB und SOURCE, die sich unter dem OS/2-Unterverzeichnis befinden, unter dem Standardverzeichnis MSC510.*

▶▶ *Bild 3: Eine universelle MAKE-Datei für ein kombiniertes OS/2-Programm.*




```
#file: hello
#generic OS/2 MAKE file for producing 'bound' executables
#
#Currently set for making HELLO.C
#Usage: C>make hello

INCLUDE=\os2\include
LIB=\os2\lib
COPT=/Lp /Fb /W3 /Zpe /G2 /Ox /I$(INCLUDE)
#COPT=/Lp /Fb /W3 /Zpie /G2 /Od /I$(INCLUDE)

hello.exe: hello.c hello
    cl $(COPT) hello.c /link /noe
#end
```

MS-C 5.1 bietet die Möglichkeit DOS, OS/2 und kombinierte Programme, die unter beiden Betriebssystemen laufen, zu erzeugen. Ein großer Vorzug der Installation sind die kombinierten Bibliotheken, wenn Sie den Compiler unter OS/2 installieren. Frühere Versionen des Compilers beinhalten eine Unmenge von Bibliotheken, einschließlich dreier Fließkomma- und einer Grafikbibliothek. Wenn Sie zu diesen Bibliotheken Versionen für jedes Speichermodell hinzufügen und zusätzlich die Versionen für OS/2, brauchen Sie außergewöhnlich viel Plattenspeicher. Sie können aber für jedes Speichermodell die Grundbibliothek mit der Grafikbibliothek (wenn gewünscht) und der Fließkommabibliothek verbinden. Dies erzeugt nur eine Bibliothek je Speichermodell. Das Installationsprogramm erledigt das automatisch und löscht auf Wunsch die nicht mehr benötigten Einzelkomponenten.

Weiterhin läßt sich ein kombinierter (Bound) Compiler erzeugen. Diese Version läuft dann unter DOS und OS/2. Das Installationsprogramm erzeugt die Datei BINDC.CMD (BINDC.BAT, wenn Sie den Compiler unter MS-DOS installiert haben), die Sie ausführen müssen, um die kombinierte Version zu erzeugen. Wir erörtern das Herstellen von kombinierten Programmen später. Sie können sich aber vorerst merken, daß eine kombinierte Applikation nur eine Untermenge der API-Aufrufe verwendet, die FAPI (Familien-Programmier-Schnittstelle) genannt wird. Dies sind Funktionen, die sowohl unter DOS als auch unter OS/2 verfügbar sind. Die Verwendung dieser Funktionen gestattet aber nicht die Verwendung der OS/2-Multitasking-Eigenschaften (da DOS diese Fähigkeiten nicht bietet). So ein Programm läuft unter beiden Betriebssystemen.

BINDC erzeugt eine kombinierte Version des Compilers in dem gewünschten Verzeichnis. Da es sich um eine kombinierte Version handelt, sollten Sie die Ausgabe von BINDC in das Verzeichnis \OS2\BIN erzeugen lassen, und die Originalkopien der Compiler löschen. So benötigen Sie nicht mehr, und diese brauchen nur Speicherplatz auf der Platte. Die kombinierte Version ist größer als das Original, aber diese eine Version ist einfacher zu handhaben.

Das Installationsprogramm erzeugt zwei Dateien: NEW-VARS.CMD und NEW-CONF.SYS. Die Datei NEW-VARS.CMD enthält die Environment-Variablen, die zur Laufzeit des Compilers aktiv sein sollten. Diese basieren auf der von Ihnen bei der Installation gewählten Verzeichnisstruktur. Sie können diese entweder in die Datei STARTUP.CMD übernehmen, oder in die Kommandodatei, die beim Einrichten der C-Entwicklungssitzung im OS/2-Programmstarter aktiviert wird. NEW-CONF.SYS enthält Zusätze für die Datei CONFIG.SYS (CONFIG.OS2 in älteren Systemen mit Dual-Boot-Option).

Neue Compiler-Optionen

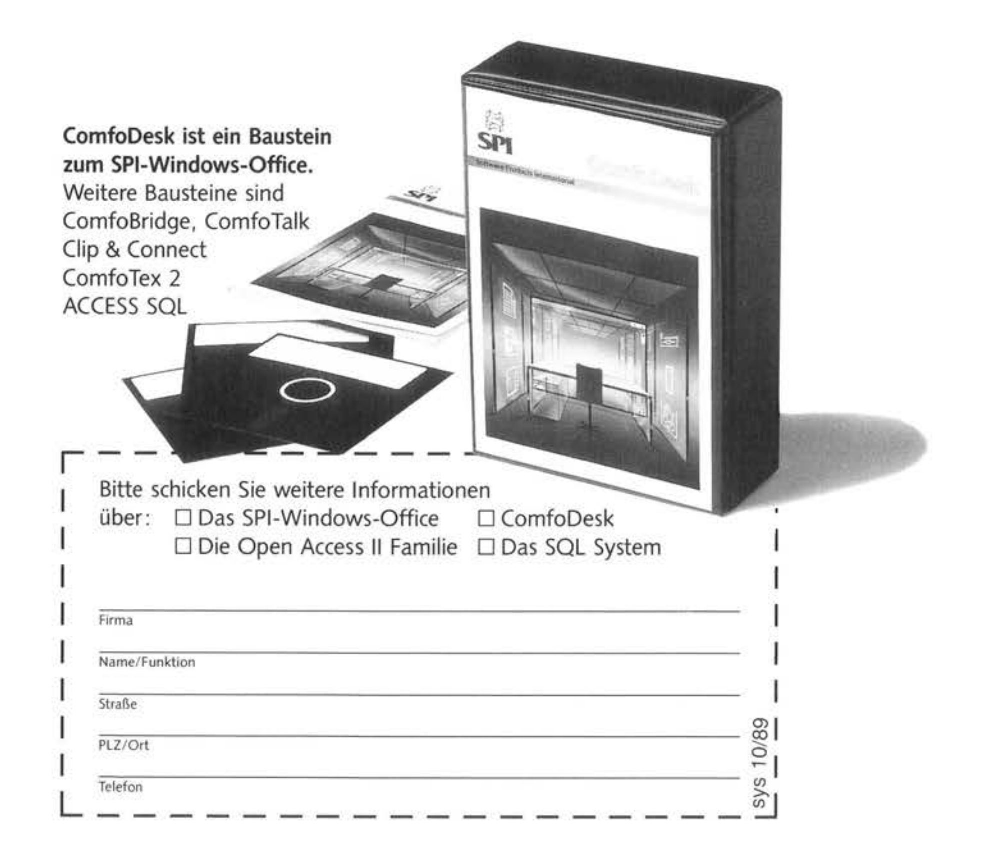
Es existieren für die OS/2-Programmierung drei Kommandozeilenoptionen. Die Option /Lr generiert ein Programm im nicht geschützten Modus (in früheren Versionen des Compilers üblich). Die Option /Lp jedoch veranlaßt den Compiler und den Linker, das Programm für den geschützten Modus zu erzeugen. Wenn Sie diese Option benützen, wird die geschützte Version der Bibliothek verwendet. Benötigt zum Beispiel die Übersetzung im nicht geschützten Modus mit kleinem Speichermodell die Bibliothek SLIBCE.LIB (die kombinierte Bibliothek im kleinen Speichermodell mit Fließkomma-Emulation), so bindet der Linker beim Aktivieren des Compiler-Schalters /Lp die Bibliothek SLIBCEP.LIB (die Version der Bibliothek für den geschützten Modus).

Die dritte Option /Fb weist den Linker an, das Programm BIND.EXE zu aktivieren, nachdem der Linkvorgang beendet ist. BIND ist ein Hilfsprogramm, das Programme im geschützten Modus in kombinierte Applikationen ändert, die dann unter beiden Betriebssystemen laufen können. Mit diesen Optionen und der Verzeichnisstruktur von vorhin können wir eine allgemeine MAKE-Datei erzeugen. Diese verwenden wir dann für unser erstes OS/2-Programm (*Bild 3*).

Die MAKE-Datei, die wir HELLO nennen (da sie für die Übersetzung von HELLO.C benötigt wird, bildet die Basis für die MAKE-Dateien für andere Programme. Sie setzt die Environment-Variablen INCLUDE und LIB für den Compiler und übergibt CL eine Reihe von Optionen. Die MAKE-Datei verwendet sowohl /Lp als auch /Fb, um eine kombinierte Version erzeugen zu lassen.

Ordnung ist das halbe Büroleben!

Lange genug diente der PC nur den Fachleuten als Werkzeug. Wenn Sie aber jetzt den Computer als leistungsfähiges Bürosystem nutzen wollen, können Sie mit ComfoDesk Ihre gewohnte Büroumgebung auf Ihrem PC installieren. Mit Schränken und Ordnern und mit einem Materiallager, in dem ein Vorrat an Briefbögen, Kalkulationsblättern usw. lagert. Ihr Taschenrechner und Terminkalender liegen griffbereit auf dem Tisch, wie Sie es gewohnt sind, ganz nach Ihrem persönlichen Arbeitsstil. ComfoDesk wird natürlich auch im Netz eingesetzt: Dabei nutzen mehrere Anwender ein gemeinsames Ablagesystem und einheitliche Formulare zur Erstellung von Schriftverkehr. Und jeder kann sich bei Bedarf Unterlagen von Kollegen ansehen, es sei denn, das Dokument wurde als privat gekennzeichnet. So entsteht Transparenz im Büro, die auch vom Computerlaien sofort durchschaut wird. Das ganze Büroleben wird leichter!



ComfoDesk ist ein Baustein zum SPI-Windows-Office.
Weitere Bausteine sind
ComfoBridge, ComfoTalk
Clip & Connect
ComfoTex 2
ACCESS SQL

Bitte schicken Sie weitere Informationen über:
☐ Das SPI-Windows-Office ☐ ComfoDesk
☐ Die Open Access II Familie ☐ Das SQL System

Firma

Name/Funktion

Straße

PLZ/Ort

Telefon

sys 10/89





SOFTWARE PRODUCTS INTERNATIONAL

SPI Software Products International (Deutschland) GmbH
Stefan-George-Ring 22+24, 8000 München 81
Tel.: 089/930090-0, Fax: 089/930090-11

Die Option /W3 setzt die Ebene 3 für Warnungen des Compilers. Dies ist die höchste verfügbare Stufe. Diese Option ist für die strenge Typüberprüfung der verschiedenen Objekte und Argumente der OS/2-Typdefinitionen sehr nützlich. Die Option /Zpe beinhaltet die beiden Optionen /Zp und /Ze. Die erste läßt den Compiler gepackte Strukturen erzeugen, die auf Bytegrenze ausgerichtet sind (im Gegensatz zur Wortausrichtung). Die zweite erlaubt Erweiterungen in C, wie die Schlüsselworte far, near und pascal, die in OS/2-Programmen häufig benötigt werden. Die /Ox-Option setzt die höchste Optimierungsstufe. Die Option /I spezifiziert dem Compiler das für die Includedateien zu verwendende Verzeichnis.

Die /G2-Option ist für das Erzeugen von 80286-Befehlen zuständig. Diese Option ist sehr nützlich, da die 80286-Befehle effizienter sind als einfache 8086-Anweisungen. Dies beschränkt die Verwendung des Programms aber auf einen 80286 oder höher, was in der OS/2-Umgebung aber sowieso der Fall ist. Probleme gibt es nur, wenn die kombinierte Applikation unter DOS auf einem 8088 oder 8086 ausgeführt wird.

Zu dieser MAKE-Datei sollte ich noch einige Dinge bemerken. Zur Fehlersuche wollen wir die symbolischen Informationen mit den Optionen /Zi und /Od erzeugen. Deshalb habe ich eine zusätzliche Zeile mit Optionen für die Fehlersuche aufgenommen. Sie können diese Optionen aktivieren oder deaktivieren, indem Sie an die erste Position einer der beiden Zeilen ein # schreiben (# bedeutet in der MAKE-Datei einen Kommentar). Beachten Sie auch die Option /NOE, die der Compiler an den Linker übergibt. Sie verhindert die erweiterte Bibliothekssuche des Linkers, was gewährleistet, daß die richtigen Bibliotheksfunktionen mit dem Programm gelinkt werden.

OS/2-Systemaufrufe

OS/2-Systemaufrufe sind den Programmen durch das OS/2-API zugänglich. Im Gegensatz zu den Interrupt-Aufrufen von DOS, sind diese Funktionen durch Systemfunktionen verfügbar (auch als Aufruftore bekannt). Das DOS-System ist auf 256 Funktionen beschränkt (durch den Interrupt 21H). Für OS/2 bestehen keine Beschränkungen für die Zahl der in der Zukunft hinzuzufügenden Funktionen. Im Gegensatz zu DOS, das Register für die Parameterübergabe benötigte, werden OS/2 die Parameter auf dem Stack übergeben. Alle OS/2-Aufrufe benützen dasselbe Format. Gibt ein Systemaufruf einen Fehlercode zurück, so ist dieser Wert 0, falls keine Fehler vorliegen, und ein vorzeichenloser Wert ungleich 0 wird im Fehlerfalle zurückgegeben. Zusätzlich arbeiten die FAPI-Funktionen wie oben schon erwähnt sowohl unter DOS als auch unter OS/2, was Ihnen das Erzeugen von kombinierten Applikationen erlaubt, die unter beiden Umgebungen laufen.

Wenn Sie einen API-Funktionsaufruf in Ihrem Programm verwenden, so fügt der Linker den Funktionscode nicht zum ausführbaren Programm hinzu, wie dies unter DOS der Fall wäre. Stattdessen werden Befehle hinzugefügt, die den Funktionscode aus der zugehörigen dynamischen Linkbibliothek (DLL) aufrufen. Wird das Programm ausgeführt, so lädt OS/2 die zugehörigen DLL's (falls sie nicht bereits im Speicher sind). So sind die Funktionen zur Laufzeit für das Programm verfügbar.

Auch wenn mehr als ein Applikationsprogramm eine API-Funktion benötigt, so befindet sich diese nur einmal im Speicher. Für eine FAPI-Funktion in einem kombinierten Programm bindet der Linker sowohl die DLL-Aufrufbefehle als auch die ausführbaren Befehle für den nicht geschützten Modus. Die ersten werden benötigt, wenn das Programm im geschützten Modus von OS/2 arbeitet, die zweiten rufen den Interrupt 21H im nicht geschützten Modus von OS/2 auf.

Da OS/2 die API-Funktionen in einer DLL aufruft, sind diese in einem anderen Segment als das aufrufende Programm. Deshalb müssen diese mit einem Far-Aufruf aufgerufen werden. Es gibt vier Arten von Parameter, die den API-Aufrufen übergeben werden können: Byte (oder Zeichen), Wort (oder vorzeichenlos), Doppelwort (vorzeichenlose Langzahl) und Adressen (Far-Zeiger). Benötigt der Aufruf nur den Wert des Parameters, so wird eine Kopie übergeben (Call by Value). Benötigt der Aufruf einen Zeiger auf den Wert, so wird die Adresse übergeben (Call by Reference). Da die Aufrufe Far-Aufrufe sind, müssen auch die übergebenen Adressen far sein. Um sicherzustellen, daß Sie der Systemfunktion eine Far-Adresse übergeben, sollten Sie das Objekt als far deklarieren, einen Cast verwenden, oder im Large-Speichermodell übersetzen. Der Einsatz der OS/2-Objektdefinitionen in den neuen Headerdateien (unten beschrieben) behebt dieses Problem.

Die Segmentadressen, die in den Far-Adressen enthalten sind, sind in Wirklichkeit Selektoren. Obwohl die Adressen im geschützten OS/2-Modus dieselbe Form Segment:Offset haben wie unter DOS, benötigt der geschützte Modus einen Selektor. Ein Selektor ist ein Index in eine Tabelle von Segmentadressen, und ist unabhängig vom physikalischen Segment. Selektoren müssen verwendet werden, da die virtuelle Speicherverwaltung von OS/2 das physikalische Segment im Speicher verschieben oder auf die Platte auslagern kann. Diese Abstraktionsebene gestattet Ihnen die Adressierung eines Far-Objekts, ohne zu wissen, welche physikalische Adresse es besitzt – oder ob es überhaupt im Speicher ist (OS/2 erledigt dies für Sie).

API-Funktionen verwenden die Pascal-Aufruffolge. Diese Konvention schreibt vor, daß die An-

OS2.H	Immer in Ihr Programm einlesen
OS2DEF.H	Gemeinsame Definitionen
BSE.H	Basisdefinitionen, liest die folgenden Dateien ein:
BSEDOS.H	Kernfunktionen
BSESUB.H	Kbd, Vio, Mou Definitionen
BSEERR.H	Fehlermakros

zahl der Parameter konstant ist, und daß diese von links nach rechts auf dem Stack abgelegt werden (in der Reihenfolge, wie sie in der Quelldatei stehen). Zusätzlich braucht bei der Pascal-Aufruffolge der Aufrufer den Stack nicht berichtigen. Daraus ergeben sich kleinere, schnellere Funktionsaufrufe. Das ist die Umkehrung der gewohnten C-Aufrufe, wobei die Argumente von rechts nach links (mit der Möglichkeit einer variablen Anzahl) auf dem Stack abgelegt werden. Hierbei muß der Aufrufer den Stack wieder in Ordnung bringen, und es ergibt sich ein langsameres, größeres Programm. Die unten beschriebenen OS/2-Headerdateien deklarieren die API-Funktionen als far pascal. Sie brauchen also nur die entsprechenden Headerdateien in Ihr Programm einzulesen. Die Aufruffolge far pascal ist in den OS/2-Headerdateien als APIENTRY definiert.

Zusätzlich verwenden die API-Funktionen die Microsoft Windows-Namenkonventionen mit bezeichnenden zwei- bis dreiphrasigen Namen in Groß- und Kleinschreibung. Tastaturfunktionen beginnen mit Kbd, Maus und Videofunktionen beginnen mit Mou und Vio. Die Namen der verbleibenden Systemaufrufe (OS/2-Kern) beginnen mit Dos. Einige kurze Beispiele: DosRead, DosCreateThread, KbdCharIn und VioWrtTTY.

Die Dokumentation der OS/2-API-Funktionen befindet sich im OS/2-Referenzhandbuch für Programmierer. Dieses Handbuch ist im Programmer's Toolkit und im Microsoft OS/2 Software Development Kit enthalten.

Headerdateien

Microsoft C 5.1 liefert sechs neue Headerdateien, von denen einige oder alle in einem Programm, das OS/2 API-Aufrufe durchführt, eingelesen werden müssen. Diese Headerdateien (in *Bild 4* gezeigt) liefern die API-Deklarationen, Funktions-Prototypen, Makros, Konstanten und Typdefinitionen, die für ein OS/2 C-Programm benötigt werden. Vor der Programmierung unter OS/2 sollten Sie mit diesen vertraut sein. Sie werden sehen, daß die meisten Typdefinitionen und Strukturen eine ähnliche Namensgebung verwenden wie die Systemaufrufe. Da viele Systemaufrufe Rückgabewerte in diese Strukturen schreiben, gestaltet sich die OS/2-Programmierung erheblich einfacher, wenn Sie diese vorher kennen.

Definieren Sie dieses Makro:	Um diese Definitionen/ Deklarationen einzuschließen:
INCL_BASE	Alle Funktionen
INCL_DOS	Kernfunktionen
INCL_SUB	Untersysteme (Kbd, Vio, Mou)
INCL_DOSERRORS	Fehlermakros
INCL_DOSPROCESS	Prozeß- und Threadfunktionen
INCL_DOSINFOSEG	Informationssegment-Aufrufe
INCL_DOSFILEMGR	Dateimanagerfunktionen
INCL_DOSMEMMGR	Speicherverwaltungsfunktionen
INCL_DOSSEMAPHORES	Semaphorenfunktionen
INCL_DOSDATETIME	Datum/Uhrzeit- und Timer-Funktionen
INCL_DOSMODULEMGR	Modul Management-Funktionen
INCL_DOSNLS	Nationale Sprachfunktionen
INCL_DOSSIGNALS	Signalfunktionen
INCL_DOSMONITORS	Monitorfunktionen
INCL_DOSSESMGR	Session Manager-Funktionen
INCL_DOSDEVICES	Device- und IOPL-Unterstützung
INCL_DOSQUEUES	Queue-Funktionen
INCL_RESOURCES	Ressource Unterstützungsfunktionen

◄◄ *Bild 4: OS/2-Headerdateien für die Programmentwicklung*

◄ *Bild 5: OS/2-Headerdatei-Kontrollmakros*

Die OS/2-Headerdateien sind hierarchisch geschachtelt. Deshalb brauchen Sie in den meisten Fällen nur OS2.H in Ihre Datei einzulesen. Die restlichen Headerdateien und Definitionen können durch verschiedene Kombinationen von Kontrollmakros (in *Bild 5* dargestellt) eingelesen werden. Diese sollten in Ihrem Programm vor der Anweisung include "OS2.H" stehen. Dies ist hilfreich, wenn Sie in Ihrem Programm nur einen Teil der API-Funktionen verwenden, da die Headerdateien groß sind und die Übersetzung schneller ausgeführt wird, wenn Sie nur einlesen, was benötigt wird.

OS2.H liest zwei Headerdateien ein. Die erste Datei, OS2DEF.H, enthält die am häufigsten benutzten Definitionen, Typedef-Anweisungen, Makros, Konstanten und Strukturen. Die zweite, BSE.H, enthält indirekt die Basisdefinitionen für die verschiedenen OS/2-Untersysteme (Tastatur, Video, Maus und DOS), und Fehlerbehandlungsmakros, indem drei weitere Headerdateien eingelesen werden: BSEDOS.H, BSESUB.H und BSEERR.H.

BSEDOS.H enthält die Definitionen, die beim Gebrauch der OS/2-Kernfunktionen benötigt werden, und kann durch die Definition von INCL_DOS vor der Anweisung #include für OS2.H in Ihr Programm eingelesen werden. BSESUB.H enthält alle Definitionen, die beim Benutzen der OS/2-Untersysteme (Kbd, Mou oder Vio) benötigt werden. Sie wird durch die Definition von INCL_SUB eingelesen. BSEERR.H enthält alle fehlerbezogenen Makros und wird durch die Definition von INCL_DOSERRORS eingelesen. Wenn es notwendig ist, so können Sie alle API-Deklarationen und Definitionen durch die Definition INCL_BASE in Ihr Programm einlesen:

```
#define  INCL_BASE
#include <OS2.H>
```

▶ *Bild 6: Eine kombinierte Version von HELLO.C für OS/2*

```
/* hello.c RHS 10/14/88
 *
 * 1988 OS/2 version of      K&R's hello.c
 */

#define    INCL_SUB          /* for Vio calls used */
#define    INCL_DOS          /* for DosExit call used */
#include   <stdio.h>
#include   <string.h>
#include   <os2.h>

void main(void);          /* function prototype */

void main(void)
    {
    char *hello_str = "Hello, world!\r\n";
    int   len = strlen(hello_str);

    VioWrtTTy(hello_str,len,0);     /* print the message */
    DosExit(EXIT_PROCESS,0);        /* exit the program */
    }
```

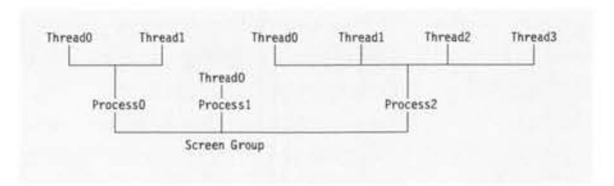
Viele gemeinsam benutzte Komponenten sind vordefiniert, es sei denn, Sie verwenden INCL_NOCOMMON in Ihrem Programm. Sie können auch spezielle Kernaufruf Komponenten durch die Definition der Makros im *Bild 5* einlesen.

Das erste OS/2-Programm

Nach der erfolgreichen Installation des Compilers können Sie beginnen, ein OS/2-Programm zu schreiben. *Bild 6* zeigt die Quelldatei für eine OS/2-Version des berühmten Programms von Dennis Ritchie (HELLO.C). Beim ersten Anblick erscheint ein OS/2-Programm wie dieses etwas groß. Dies ist nicht die bekannte einfache Kernighan/Ritchie-Version. Das Programm erfüllt aber denselben Zweck. Es erlaubt uns, das API von OS/2 zu erkunden, und gestattet uns, einige Headerdateien, Definitionen und Funktionen zu verwenden. Außerdem erlaubt es uns, das erste OS/2-Programm zu schreiben.

Auf den zweiten Blick erkennen wir die bekannte Struktur, die wir so oft zum Schreiben wartbarer, effizienter C-Programme verwendet haben. Sie sehen die Verwendung von INCL_SUB und INCL_DOS, um die Funktionsprototypen für den einzigen Vio-Untersystemaufruf und den einzigen Kernaufruf einzulesen. Der Printf-Aufruf wurde durch die Funktion VioWrtTTY ersetzt, die einen String auf den von unserem Programm verwendeten logischen Bildschirm ausgibt (alle Video-Ausgaben in OS/2 werden über logische Bildschirme abgewickelt). Diese Funktion benötigt sowohl den String als auch die Länge als Parameter und beherrscht ebenso die C-Escape-Sequenzen \r und \n, um eine Carriage-Return-Kombination zu erzeugen. Alle Vio-Aufrufe benötigen eine 0 als letzten Parameter.

Nach der Ausgabe des Strings beendet DosExit das Programm. Der erste Parameter gibt an, ob der ganze Prozeß oder nur dieser Thread beendet werden soll. Der zweite Parameter ist der Exitcode, der an den aufrufenden Prozeß übergeben wird, und der identisch dem an exit, der traditionellen C-Standard-Bibliotheksfunktion, übergebenen ist.



▶▶ *Bild 7: Die Multitasking-Fähigkeit ist der größte Unterschied zu DOS. Die kleinste Ausführungseinheit ist der Thread. Jeder Prozeß besteht aus mindestens einem Thread. Alle hier gezeigten Prozesse teilen sich den gleichen Bildschirm und die gleiche Tastatur. Sie sind Teil derselben Bildschirmgruppe. Benötigt ein Prozeß eine andere Bildschirm-/Tastatur-Kombination, so muß er in einer anderen Bildschirmgruppe gestartet werden.*

Dieses Programm kann übersetzt, gelinkt und mit BOUND bearbeitet werden, um eine kombinierte Applikation zu erzeugen. Dasselbe Programm läuft dann unter OS/2 oder DOS. Die MAKE-Datei für dieses Programm sehen Sie in *Bild 3*. Wenn Sie noch kein C-Programm mit API-Funktionen geschrieben haben, so empfehle ich, dieses Programm zu übersetzen und laufen zu lassen. Es hilft sicher, etwas vom Geheimnis um OS/2 zu nehmen, und bereitet Sie auf den nächsten Schritt vor: die Welt der Multithread-Programmierung.

Mehrere Threads

Der größte Unterschied zwischen OS/2 und DOS ist die Multitasking-Fähigkeit von OS/2. Multitasking steigert die Effizienz der meisten Programme sehr stark, wenn sie dafür ausgelegt sind. Aber OS/2 erlaubt nicht nur das Ausführen von mehreren Programmen, sondern auch das gleichzeitige Ausführen desselben Programms. Das Multitasking Konzept von OS/2 sehen Sie in *Bild 7*. Es basiert auf Threads, Prozessen und Bildschirmgruppen. Der Thread ist die kleinste Einheit, die getrennt ausgeführt wird. Threads sind in einem Prozeß zusammengefaßt, der die Ressourcen, wie Dateien, Speicher und Threads kontrolliert. Ein Prozeß besitzt zumindest einen, und maximal 255 Threads. Der Hauptthread ist der Teil des Programms, der am Anfang ausgeführt wird. Ein Prozeß kann einen Thread zum Verwalten einer Ressource verwenden, aber ein Thread besitzt diese Ressource nicht. Ein Thread erbt die Umgebung (offene Dateien und so weiter) und teilt die Code- und Datensegmente mit seinem Vaterprozeß. Die Prozesse, die logische Tastatur und logischen Bildschirm teilen, gehören zur selben Bildschirmgruppe.

Das Betriebssystem erlaubt Multitasking auf allen drei Ebenen (Bildschirmgruppen, Prozesse und Threads). In diesem Artikel beschäftigen wir uns mit der gleichzeitigen Ausführung von Threads im selben Prozeß. Programme dieser Art nennen wir Multithread-Programme.

Die Ausführung mehrerer Threads

OS/2 verteilt die CPU-Zeit auf mehrere Threads durch einen zeitscheiben- und prioritätsgesteuerten Scheduler (spätere Versionen nutzen die Vorteile von Rechnern mit mehreren Prozessoren). In Wirklichkeit wird nur jeweils ein Thread aus-

geführt, aber die CPU wird schnell von einem Thread auf den anderen umgeschaltet, so daß es scheint, als würden sie alle gleichzeitig ausgeführt – solange die Applikationen, die aus mehreren Threads bestehen, nicht die Systemressourcen und die CPU mißbrauchen.

Der Task-Scheduler kontrolliert, welcher Thread wieviele Zeitscheiben erhält. Die Priorität des Threads bestimmt den Zugriff auf die CPU (bei einer höheren Priorität, bekommt der Thread relativ zu anderen Threads mehr CPU-Zeit). Deshalb kann ein vorher untätiger Thread mit einer hohen Priorität, der plötzlich bereit ist, einem gerade laufenden Thread mit niedrigerer Priorität die CPU entziehen. Andererseits muß ein Thread mit niedrigerer Priorität warten, bis alle Threads mit einer höheren Priorität untätig sind, bevor der Scheduler ihm CPU-Zeit gewährt.

Der OS/2 Task-Scheduler kennt drei Kategorien von Prioritäten. Die höchste Priorität ist zeitkritisch. Sie soll von Tasks verwendet werden, die auf Ereignisse reagieren müssen (wie Kommunikationsprogramme oder Eingaben). Die nächste Kategorie ist normal. Ein neuer Thread erhält diese automatisch, und die meisten Threads sollten sie auch beibehalten. Die letzte Kategorie ist untätig. Diese Prioritätsstufe sollten Threads erhalten, die nur laufen, wenn kein Thread mit höherer Priorität zum Laufen bereit ist (etwa Druckerspooler). In jeder Kategorie gibt es 32 Stufen. Die voreingestellte Priorität eines Vordergrund-Prozesses ist normal Stufe 0. Prozesse, die im Hintergrund laufen, gehören auch der normalen Stufe an, besitzen aber eine niedrigere Priorität. Generell erhalten Vordergrund-Tasks höhere Priorität als Hintergrund-Tasks.

Der Task-Scheduler läßt immer den Thread mit der höchsten Priorität laufen, der bereit ist. Sind zwei oder mehrere Threads mit der höchsten Priorität bereit, so gibt ihnen der Scheduler nach einem »Round-Robin«-Verfahren die CPU. Wird ein Thread blockiert (wenn er auf ein Ereignis wartet), so unterbricht ihn OS/2 und läßt einen anderen Thread laufen. Das Statement TIMESLICE= in der Datei CONFIG.SYS kontrolliert die minimale und die maximale Zeitscheiben-Zeit von OS/2.

Wie wir schon wissen, wird ein Thread blokkiert, wenn er auf ein Ereignis wartet. Ein Thread läuft, wenn er CPU-Zeitscheiben erhält. Ist ein Thread nicht mehr blockiert, bekam aber noch keine CPU-Zeitscheiben, so sagt man, er ist bereit zur Ausführung.

Die Planung eines Multithread-Programms

Ein Prozeß kann aus mehreren Threads bestehen. Mehrere Threads können die Ressourcen der Maschine effizienter kontrollieren als eine Applikation aus einem Thread. Zum Beispiel können Ausdrucke, Kommunikation, Dateiübertragungen und das Sortieren einer Datenbank gleichzeitig durch mehrere Threads erledigt werden, während der Hauptthread den Anwender bedient. In der Multitasking-Umgebung von OS/2 ist die Multitasking-Architektur eines Programms wichtig, um sicherzustellen, daß keine einzelne Task die Maschinenressourcen vergeudet. Gordon Letwin, der Architekt von OS/2, entdeckte zuerst diesen Zusammenhang für das gemeinsame Benutzen der Ressourcen: Programme müssen die Regeln befolgen, um zusammenzuarbeiten. Das Befolgen vermeidet die Suche nach dem Übeltäter, wenn ein Programm die Umgebung mißbraucht, und gestattet dem System, mit der größtmöglichen Performance zu arbeiten.

Deshalb bedeutet die Planung einer Applikation mit mehreren Threads, daß jede Aufgabe, die das Programm erledigt, am besten mit mehreren Threads zu realisieren ist. Es gibt Hilfen bei der Planung eines solchen Programms: Nehmen Sie nie an, daß OS/2 ein Programm oder eine Routine in einer bestimmten Weise ausführt. Dies bedeutet im einzelnen:

- Nehmen Sie nicht an, daß eine Routine vor einer anderen ausgeführt wird.
- Nehmen Sie nicht an, daß eine Routine eine bestimmte Anzahl von Millisekunden ausgeführt wird.
- Nehmen Sie nicht an, daß spätere Versionen von OS/2 Tasks in der gleichen Weise ablaufen lassen, wie die aktuelle.
- Nehmen Sie nicht an, daß verschiedene Threads um die CPU ringen. Spätere Versionen von OS/2 können auf Multiprozessor-Systemen laufen, wo dann mehrere Threads wirklich parallel laufen. Obwohl wir wissen, daß sie derzeit nicht parallel laufen, denken Sie doch daran.
- Es gibt keine direkte Zuordnung zwischen CPU-Zeitscheiben und CPU-Zyklen.
- OS/2 garantiert keine Thread-Reihenfolge während der Ausführung. Obwohl der Hauptthread immer der erste ist, der ausgeführt wird, gibt es ab dem Start des zweiten Threads keine Garantie für die Reihenfolge mehr.

Der letzte Punkt bedeutet aber nicht, daß es keine Kontrollmöglichkeiten für die Manipulation von Ereignissen oder den seriellen Zugriff auf Ressourcen zwischen mehreren Threads gibt. Hierbei kommen Prioritätsebenen und Semaphore ins Spiel. Diese besprechen wir später, aber Sie sollten sich schon jetzt merken: Machen Sie keine Annahmen!

Ein Multithread-Programm

Das Erzeugen eines neuen Threads ist ziemlich einfach. Nehmen wir zum Beispiel ein Programm, das alle Eingaben von der Tastatur, außer dem Escape-Zeichen, welches das Pro-

▶ *Bild 8: Ein einfaches Tastaturthread-Programm.*

▶▶ *Bild 9: DosCreateThread, dessen Funktionsprototyp hier gezeigt ist, kann neue Threads erzeugen. Der Zeiger functionptr zeigt auf die Funktion des Threads. Die ID des Threads wird an der Stelle abgelegt, auf die threadidptr zeigt, und der Thread benutzt den Stack, dessen Spitze an der Stelle ist, auf die stack zeigt.*

```
/*
 * Simple keyboard thread example
 *
 * This program illustrates how a process might start
 * a keyboard thread which will terminate the process when
 * the user presses the Esc key.
 *
 * The program starts a keyboard thread which blocks on keyboard
 * input and terminates the entire program when the user presses
 * the Esc key.  All other keys are ignored and thrown away.
 */

#define    INCL_DOS
#define    INCL_SUB

#include   <os2.h>
#include   <mt\stdio.h>
#include   <mt\process.h>

#define    ESC    0x1b
#define    TRUE    1
#define    THREADSTACK    512

char keythreadstack[THREADSTACK];

void keyboard_thread(void);
void main(void);

void main(void)
    {
    TID    threadid;

    if(DosCreateThread(keyboard_thread, &threadid,
            &keythreadstack[THREADSTACK-1]))
        exit(-1);

    while(TRUE)    /* replace this with
        ;              code for main program */
    }

void keyboard_thread(void) /* keyboard thread code */
    {
    KBDKEYINFO keyinfo;

    while(TRUE)
        {
        KbdCharIn(&keyinfo,IO_WAIT,0);  /* wait for keystroke */
        if(keyinfo.chChar == ESC)      /* if ESC pressed, break */
            break;
        }
    DosExit(EXIT_PROCESS,0);           /* terminate the process */
    }
```

gramm abbricht, ignoriert. In einem DOS-Programm gäbe es zwei mögliche Lösungen. Sie können Ihr Programm so auslegen, daß es die Tastatur abprüft, was in einer komplexen Applikation sehr mühsam ist. Ihr Programm könnte auch den BIOS Tastatur-Interrupt abfangen, und dem Hauptprogramm ein Signal senden, wenn die Escape-Taste gedrückt wird.

Unter OS/2 sind diese Lösungen nicht notwendig oder möglich. Sie können stattdessen einen Thread starten, der auf die Tastatureingabe wartet (er wird blockiert, bis ein Tastendruck ansteht). Drückt der Anwender eine Taste, so untersucht der Thread diesen Tastendruck. Handelt es sich nicht um die Escape-Taste, so wartet er weiter. Handelt es sich um die Escape-Taste, so beendet der Thread den ganzen Prozeß.

Ein Blick auf das im *Bild 8* gezeigte Programm macht diese Prozedur verständlicher. Der Haupthread beginnt da, wo alle C-Programme beginnen, mit dem Aufruf von main. Der Hauptthread erzeugt den Tastaturthread mit einem Aufruf der API-Kernfunktion DosCreateThread.

DosCreateThread benötigt mehrere Parameter, wie dies der Funktionsprototyp in *Bild 9* zeigt. Der erste Parameter ist ein Zeiger auf die Funktion des Threads. Hier ist die Funktion keyboard_thread bezeichnet. Der zweite Parameter

```
unsigned DosCreateThread(void (far *) functionptr(void),
                         TID *threadidptr, void *stack);
```

ist ein Zeiger auf die Variable, in der OS/2 den Bezeichner des Threads ablegt, nachdem dieser erfolgreich erzeugt wurde. Der letzte Parameter ist die Adresse des Stacktops, des für den Thread allokierten Stacks, der mindestens 512 Byte groß sein sollte. Um die Adresse des Stacktops von keythreadstack zu übergeben, müssen wir die Adresse des letzten Bytes von keythreadstack in der gezeigten Weise übergeben.

Alle API-Funktionen geben im Fehlerfalle ungleich 0 zurück. Liefert DosCreateThread 0, so wissen wir, daß der Thread gestartet ist. Der neu erzeugte Thread führt sofort das in keyboard_thread enthaltene Programm aus. Obwohl der Aufruf von DosCreateThread überall im Programm stehen kann, gewährleistet ein frühes Aufrufen den Programmabbruch auch, wenn der Anwender am Anfang die Escape-Taste drückt. Die Schleife while(TRUE) kann durch den Programmteil ersetzt werden, den Sie benötigen.

Sehen wir uns jetzt die Funktion keyboard_thread an. Der Programmteil eines Threads sollte immer in einer Funktion enthalten sein. Sie können die Programmteile für diese Funktion nicht in main oder eine andere Funktion einfügen. Sie können eine Threadfunktion auch nicht direkt aufrufen. Threadfunktionen sind eigentlich eine Erweiterung von OS/2 in C.

Keyboard_thread startet die Ausführung unmittelbar nach dem Erzeugen, und ruft dann in einer Schleife die Kbd-Untersystemfunktion KbdCharIn auf. Der erste Parameter ist ein Zeiger auf die OS/2-Struktur KBDKEYINFO (die in einer weiteren Folge dieser Serie näher erörtert wird), die nach der Funktionsrückkehr Informationen über die gedrückte Taste enthält. Das Element chChar enthält den ASCII-Wert der gedrückten Taste. Der zweite Parameter gibt an, wie lange die Funktion warten soll, bis der Benutzer eine Taste drückt. IO_WAIT zum Beispiel veranlaßt die Funktion unendlich auf den Tastendruck zu warten. OS/2 unterbricht den aufrufenden Thread bis die Bildschirmgruppe dieses Threads einen Tastendruck vom Benutzer empfängt.

Nach dem Tastendruck, weckt OS/2 den Thread und kehrt vom Aufruf KbdCharIn zurück. Der Thread untersucht dann den Tastendruck, und beendet die Schleife, wenn es sich um die Escape-Taste gehandelt hat, was den Aufruf von DosExit und damit das Ende des gesamten Programms bedeutet. Beachten Sie, daß der Gedanke hinter der Funktion ähnlich der objektorientierten Programmierung ist. Die Funktion keyboard_thread enthält die Kontrolle über die Tastatur und den Programmabbruch. Das Hauptprogramm muß nicht wissen, wie es funktioniert, und was es erledigt – dies erledigt der Thread ganz alleine.

```
#
# make file for key.c example found in Figure 8
#

INCLUDE=\os2\include\mt
LIB=\os2\lib
COPT=/Lp /W3 /Zp /Zie /Zl /G2s /I$(INCLUDE)  /Alfw

key.exe: key.c key
    cl $(COPT) key.c /link /co llibcmt
```

◄◄ *Bild 10: MAKE-Datei für das einfache Tastaturthread-Programm.*

```
abs      labs      memset    strcmpi   strnset
atoi     lfind     mkdir     strcpy    strrchr
atol     lsearch   movedata  stricmp   strrev
bsearch  memccpy   putch     strlen    strset
chdir    memchr    rmdir     strlwr    strstr
getpip   memcmp    segread   strncat   strupr
halloc   memcpy    strcat    strncmp   swab
hfree    memicmp   strchr    strnicmp  tolower
itoa     memcmove  strcmp    strncpy   toupper
```

◄ *Bild 12: Diese Funktionen der Standard-C-Bibliothek sind wiedereintrittsfest und können daher von mehr als einem Thread gleichzeitig benutzt werden.*

Der Hauptthread eines Programms muß am Leben bleiben, bis alle anderen Threads sich beendet oder ihre Aufgaben erledigt haben. Stirbt der Hauptthread, so sterben auch alle anderen Threads. Wenn Sie in den Hauptthread-Programmteile einfügen, die ihn beenden, so stirbt der Tastaturthread (und alle anderen Threads) auch. Erreicht ein anderer Thread sein Programmende, ohne eine Beendigungsroutine aufzurufen, so stirbt er, beeinflußt aber die anderen Threads nicht. Threads können sich explizit selber durch den Funktionsaufruf DosExit beenden:

```
DosExit(EXIT_THREAD, term_code);
```

Oder der Thread kann den ganzen Prozeß beenden:

```
DosExit(EXIT_PROCESS, term_code);
```

Die MAKE-Datei für dieses Beispiel sehen Sie im *Bild 10*. Die Optionen /Gs und /G2 werden benutzt, um die Stacküberprüfung zur Laufzeit zu deaktivieren. Dieser würde zur Laufzeit einen falschen Stacküberlauf beim Thread melden. Wollen Sie dies nur auf den Programmteil des Threads beschränken, so können Sie ein #pragma einfügen:

```
#pragma check_stack(off)
/* Thread Funktion */
#pragma check_stack(on)
```

Hier wird die Stacküberprüfung zur Laufzeit zwischen den beiden #pragma-Anweisungen abgeschaltet.

Muß der Hauptthread Dateien schließen, oder einen anderen Prozeß oder Ressourcen beenden, so kann die Threadfunktion ein Semaphor setzen (wie ein Flag), das vom Hauptthread abgeprüft wird, statt des außergewöhnlichen Abbruchs. Eine weitere Möglichkeit wäre der Gebrauch von DosExitList (*Bild 11, Seite 32*).

Wiedereintrittsfestigkeit

Es gibt einen wichtigen Punkt beim Schreiben von Programmen, die aus mehreren Threads bestehen. Das Problem tritt auf, wenn mehr als ein Thread zur gleichen Zeit versucht, ein Programmteil auszuführen. Dieses Problem gibt es mit den OS/2 API-Funktionen nicht. Diese Funktionen sind für eine Umgebung mit mehreren Tasks programmiert. Die Standard-C-Bibliothek, und Ihre eigenen Funktionen sind kritischer.

Stellen Sie sich folgendes vor: Die Funktion printf der Standardbibliothek benutzt einen internen Puffer, um die Zeichenkette aufzubereiten, die nach Standard Output ausgegeben werden. Nehmen wir an, ein Thread ist mitten in der Arbeit und hat den halben Puffer aufbereitet. Wenn der nächste Thread die Funktion printf aufruft, so wird der Puffer des ersten Threads überschrieben, und es gibt ein Chaos. Dieser Zustand ist nicht akzeptabel, und wenn Sie nicht Ihre eigenen Semaphore zur Kontrolle jeder Bibliotheksfunktion (was keine akzeptable Lösung darstellt) implementieren, erleben Sie böse Überraschungen.

Gott sei Dank, gibt es noch zwei andere Lösungen. Wenn Sie ein Programm mit mehreren Threads schreiben, so können Sie nur im Hauptthread Standard-Bibliotheksfunktionen verwenden. Dies gewährleistet, daß nur ein Thread zur selben Zeit die Funktionen der Standardbibliothek benutzt. Mit der Ausnahme einiger Funktionen ist die Standardbibliothek nicht wiedereintrittsfest, da sie für eine Ausführung mit einem Thread gedacht sind (*Bild 12*). Müssen Sie den Zugriff für eine bestimmte Funktion der Standardbibliothek (oder Ihre eigene Funktion) regeln, gibt es zwei Möglichkeiten:

- Verwenden Sie die API-Funktion DosEnterCritSec, um kurzzeitig alle anderen Threads einzufrieren. Obwohl dies funktioniert, ist es nicht die beste Lösung und hier nur der Vollständigkeit halber erwähnt. Es gibt zu viele Fehler, die gemacht werden können.
- Verwenden Sie ein Semaphor, um die Ausführung einer Funktion zu kontrollieren. Diese Lösung ist besser, da nur der Thread, der die gemeinsame Funktion aufrufen will, beeinträchtigt wird. Der Rest läuft unbeeinträchtigt weiter. DosEnterCritSec hingegen friert alle anderen Threads ein.

Können Sie nicht ohne die Standardbibliothek leben, gibt es noch eine Alternative: Die Standardbibliothek für Programme mit mehreren Threads. Während frühere Versionen des Compilers die Unterscheidung zwischen wiedereintrittsfesten und nicht wiedereintrittsfesten Funktionen verlangten, unterstützt Microsoft jetzt eine Version der Standardbibliothek, LLIBCMT.LIB, die komplett wiedereintrittsfest ist und mehrere Threads unterstützt. Wenn Sie ein Programm schreiben, das diese Bibliothek verwendet, müssen Sie die neuen Bibliotheksfunktionen _beginthread und _endthread verwenden (statt DosCreateThread und DosExit). Mehr Informationen darüber stehen im Kasten *DosCreateThread kontra _beginthread*.

▸▸ *Bild 13: Das Tastaturprogramm aus Bild 8 mit der Semaphoren-Erweiterung.*

```
#define    INCL_DOS
#define    INCL_SUB
#include<stdio.h>
#include<process.h>
#include<os2.h>

#define    ESC    0x1b
#define    TRUE 1

void keyboard_thread(void);
void main(void);

#define    THREADSTACK    512

char keythreadstack[THREADSTACK];
long CountSem =    0L;
unsigned count = 0;

void main(void)
    {
    TID    threadid;

    DosSemClear(&CountSem);

    if(DosCreateThread(keyboard_thread,&threadid,
            &keythreadstack[THREADSTACK-1]))
        exit(-1);

    while(TRUE)    /* insert code for main program here */
        {
        DosSleep(100L);
        DosSemRequest(&CountSem,-1L);
        if(count > 3)
            break;
        DosSemClear(&CountSem);
        }
    }

void keyboard_thread(void)  /* keyboard thread code */
    {
    KBDKEYINFO keyinfo;

    while(TRUE)
        {
        KbdCharIn(&keyinfo,IO_WAIT,0);  /* wait for keystroke */
        if(keyinfo.chChar == ESC)      /* if ESC pressed, break */
            {
            DosSemRequest(&CountSem,-1L);
            count++;
            DosSemClear(&CountSem);
            }
        }
    DosExit(EXIT_PROCESS,0);          /* terminate the process */
    }
```

Sie können auch Ihre eigenen Funktionen für mehrere Threads schreiben. Hierbei sind aber drei Richtlinien zu beachten. Funktionen für mehrere Threads können den Interrupt nicht sperren, oder eine INT-Anweisung ausführen. Sie sollten die Werte des Segmentregisters nicht ändern oder manipulieren. Außerdem muß der Zugang zu globalen oder statischen Daten von Funktionen, die von mehreren Threads aufgerufen werden können, streng kontrolliert werden (wie wir in der Diskussion über printf festgestellt haben). Der zu bevorzugende Weg für diese Kontrolle sind OS/2-Semaphore.

Threadkontrolle

Eine genaue Darstellung der OS/2-Semaphore finden Sie im Microsoft System Journal in der Ausgabe Juli/August '88 (Seite 40) unter dem Titel »Koordination von Threads mit Semaphoren«, aber wir wiederholen die wichtigsten Punkte aus diesem Artikel.

Obwohl OS/2 einiges an Unterstützung für die Kommunikation zwischen Prozessen bietet (Pipes, Queues, Signale und gemeinsamen Speicher), sind Semaphore die bevorzugte Methode, um mehrere Threads zu koordinieren. Sie können zum Serialisieren eines Programmteils oder einer Ressource verwendet werden, welche nicht gleichzeitig benutzt werden kann. Auch können Sie Semaphore benutzen, um einem Thread mitzuteilen, daß ein Ereignis eingetreten ist. Unter anderem bietet OS/2 RAM-Semaphore (die von Threads im selben Prozeß benutzt werden). Diese sind am einfachsten zu implementieren, weshalb wir sie in unserem ersten Programm verwenden.

MS-DOS ist ein Betriebssystem, das nur eine Task zur selben Zeit laufen lassen kann. Ein Programm, das unter DOS läuft kann die Ressourcen alleine benutzen. Dies beinhaltet auch das Sperren von Interrupts und einen ununterbrochenen Zugang zu den Ressourcen. Die Signalbehandlung unter DOS ist einfach. Eine globale Variable oder ein Flag kann benutzt werden, um verschiedene Programmteile zu koordinieren. Ein Prozeß kann warten, während das Flag gesetzt ist, bis es ein anderer löscht. Nachdem der Prozeß es gelöscht hat, setzt der wartende Prozeß das Flag, in der Sicherheit, alleine Zugriff auf die Ressourcen zu haben, was auch die Flagvariable beinhaltet.

Unter OS/2 ist diese Art der Signalübermittlung nicht möglich, da es (ohne Kontrolle) keine garantierte Reihenfolge für die Threadausführung gibt. Weiterhin kann ein Thread das Flag lesen, während ein anderer es setzt oder löscht. Oder der zweite Thread setzt das Flag in dem Augenblick, in dem der erste Thread begonnen hat, das Flag selbst zu setzen, als er unterbrochen wurde. So können mehrere Threads gleichzeitig Zugriff auf die Ressource erhalten. Das Ergebnis ist chaotisch. Weiterhin belastet das ständige Abprüfen des Flags die CPU unnötig und verlangsamt das System.

Semaphore sind eine elegante Lösung dieses Problems in der Multitasking-Umgebung von OS/2. In einem ununterbrechbaren Schritt testet und setzt ein Aufruf einer Kernfunktion das Semaphor. Deshalb kontrolliert das Semaphor den Zugang zu der Ressource und gestattet einer Task einer anderen das Übermitteln eines Ereignisses.

Um eine einfache Anwendung für Semaphore zu demonstrieren, stellen Sie sich vor, wir erweitern das Tastaturthread-Programm in *Bild 8*. Die Erweiterung veranlaßt den Tastaturthread bei jedem Tastendruck der Escape-Taste einen Zähler zu erhöhen (anstatt das Programm zu beenden). Der Hauptthread untersucht periodisch den Zähler und sobald dieser größer als ein bestimmter Wert (sagen wir 3) ist, beendet der Hauptthread das gesamte Programm.

Das Problem in der Multithread-Umgebung von OS/2 ist der serielle Zugriff auf die Ressourcen, besonders auf die Zählervariable, die von mehr als einem Thread benötigt wird. Hier werden Semaphore benötigt. Durch ein Semaphor können wir den Zugriff auf die Zählervariable

Powertools for DOS · WINDOWS · OS/2

Now at TopSpeed !

smart Linker à la Turbo-Pascal	✔
symbolischer Debugger	✔
post mortem Debugger	✔
symbolischer Disassembler	✔
ROM-fähig, 86/186-386	✔
sieben Speichermodelle	✔
dynamic linking (DLL)	✔
Programme > 640 Kbytes	✔
Windows, OS/2 und PM	✔
smart Make Utility	✔
integrierte Oberfläche	✔
10 Texte > 64 K	✔
Laufzeitsystem im Quelltext	✔
Profiler + Watcher	✔
Turbo- und MS-C kompatibel	✔
validierte ANSI-Version	✔
Mixed-Language-Development	✔
Parameterübergabe in Registern	✔

TopSpeed statt Turbo ? Dafür spricht die völlig frei konfigurierbare Bediener-Oberfläche. Oder die freie Mischbarkeit von **C und Modula-2** Routinen (Mixed Language). Auch der erzeugte **register-optimierte Code** ist eine Klasse für sich (disassemblieren und staunen!). TopSpeed-C versteht natürlich Microsoft und Turbo Sourcecode gleichermaßen.

Neue Konzepte: **dynamisches Linken** (!) auch **für DOS** (Programme endlich größer als 640 kbyte), **Windows- und OS/2-PM**- Unterstützung für geplagte Entwickler. Quelltext-Portabilität zwischen DOS und OS/2. Smarte Make Utility mit eigener Intelligenz (keine komplizierten Makefiles mehr!). Kontext sensitive **Hilfe mit Hyperlinks.** Multiple Editoren mit beliebiger Textlänge.

Eingebaut: ASCII-Table, Calculator, File Search & View mit Picklisten. Gescheiter Support durch einen erfahrenen Entwickler.

Zahlreiche Hilfsprogramme: **symbolischer Profiler,** Watch-Utility für Interrupt-Probleme, **post mortem Debugging,** und mehr. Die gesamte Laufzeitbibliothek (100% kompatibel zu Modula-2) im kommentierten Quellcode.

Validierter ANSI-Compiler mit allen ANSI-Warnungen. **Multi-Thread** Programmierung auch **unter DOS.** Interface zum BGI-Modell von Borland.

QuickStep-Tools – starke Partner

für Pascal, Modula-2 und C	✔
MS-Windows (3.0, 286, 386)	✔
DOS + OS/2-Version inkl. PM	✔
Present. Manager (DOS + OS/2)	✔
Numerik	✔
Datenbankverwaltung	✔
serielle Kommunikation	✔
Pascal -> C zu 100%	✔
Resource-Compiler	✔
deutsches Produkt und	✔
deutsche Dokumentation	✔
SAA-Standard	

Wenn Sie jemals versucht haben Windows, Menüs, Datenverwaltung und Kommunikation zu einem harmonischen Ganzen zusammen zu fügen, wissen Sie um den enormen Aufwand. Soll dabei noch die Lauffähigkeit unter mehreren Betriebssystemen garantiert sein, sind profunde Kenntnisse und viel Zeit nötig. Wir haben uns diese Zeit genommen, damit Sie mehr Zeit haben. Machen Sie Schluß mit Tools, die nicht zueinander passen oder nur DOS unterstützen! Nutzen Sie die Vorteile der aufeinander abgestimmten Komponenten der QuickStep-Tools und eröffnen Sie sich neue Perspektiven.

Presentation Manager für DOS

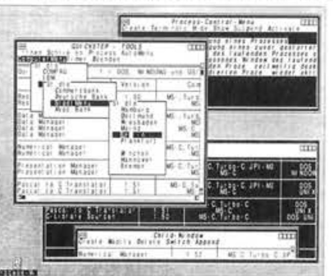
Die QuickStep-PM-Bibliothek ist kompatibel und funktional identisch zum OS/2-PM. Der QS-PM ist "event-driven" (Maus, Tastatur, Timer), arbeitet im Textmodus und ist daher sehr kompakt und schnell. Bausteine wie Menüs, Windows (Parent/Child), Accelerators, Buttons, Scrollbars, List- und Messageboxen inkl. einer vollen Farb- und Fontunterstützung (EGA, VGA) sind aufruf-, nachrichten- und funktionskompatibel implementiert.

Datenbankverwaltung

Durch mächtige HighLevel-Prozeduren schreiben Sie mit wenigen Aufrufen auch komplexe Applikationen, die beliebig viele Daten- und Index-Dateien mit variablen Datensatz- und Schlüssellängen verkraften. Datenbanken werden in der Metasprache des Resource Compilers definiert. Weitere Stärken: Passwort-, DES-, CRC-Schutz, dbase-Datei-Im/Export, Umlaute (DIN 5007), definierbare Sortierkriterien.

Numerik ?

Kein Problem mit dem Numerical Manager. Ausgesuchte Methoden wie Fourier-Transformation (FFT), reelle und komplexe Matrizen inkl. Eigenwerten/vektoren, (nicht)lineare Fitting- und Optimierungsverfahren, Differentialgleichungen, digitale Fil-



Screendump (Textmode EGA) von QS-PM

ter, Splines und komplexe Funktionen stehen für Ihre Datenanalyse zur Verfügung.

Der heiße Draht

Sie suchen ein Tool zur seriellen Kommunikation, das gepuffert und interrupt-getrieben im Hintergrund mit bis zu 115200 Baud 8 Ports gleichzeitig mit Handshake, XON/XOFF, Parity bedienen kann?
Wir haben es: den Communications Manager Turbo-Talk.

Pascal -> C

Testen Sie unseren 100% Pascal-to-C-Translator! Schicken Sie uns ein bis zu 200 Zeilen langes Turbo-Pascal Programm und urteilen Sie selbst.

QuickStep für alle

Die QuickStep-Tools sind für Turbo-, Microsoft-, und TopSpeed-C, Turbo-Pascal und TopSpeed-Modula-2 und für DOS, MS-Windows und OS/2 erhältlich.

TopSpeed Modula-2(2.0) & C

Standard Edition	DOS	498,-
Professional Edition	DOS	1198,-
Professional Edition	OS/2	1498,-
B-Tree Toolbox		498,-
Communication Toolbox		498,-

QuickStep-Tools

Presentation Manager	898,-
Data Manager	698,-
Numerical Manager	1098,-
Communications Manager	498,-
Support-Kit Comm. Manager	298,-
Pascal-to-C Translator	498,-
P2C-Bibliotheken	298,-

OS/2- und Windows-Version auf Anfrage
Preise zuzügl. DM 17,- Versandkosten (Ausland DM 30,-)

Vertrieb:
Kirchgasse 24
Tel. 06121/301065
Fax. 06121/372815

Support:
Erlkönigweg 9
Tel. 06121/42771

LAUER & WALLWITZ

►► *Bild 14: HELLO0.C, eine Multithread-Version von HELLO.C*

regeln, so daß nur ein Thread jeweils die Variable liest oder schreibt.

Das geänderte Listing sehen Sie in *Bild 13* (das Programm kann mit derselben MAKE-Datei übersetzt werden, die im Bild 10 gezeigt ist). Es wird die Semaphorvariable CountSem erzeugt, die der Hauptthread durch einen DosSemClear Funktionsaufruf löscht. Die While-Schleife wurde um das Lesen der Zählervariablen erweitert. Der Hauptthread schläft 100 Millisekunden (ungefähr drei 32 Millisekunden Zeitscheiben), und ruft dann DosSemRequest für den Zugriff auf CountSem auf. Der Parameter -1L blockiert den aufrufenden Thread bis das Semaphor gelöscht ist – daher kann auf die Zählervariable nicht zugegriffen werden, wenn der Tastaturthread bereits die Kontrolle übernommen hat.

Der Hauptthread wertet die Zählervariable aus, und beendet die Schleife, wenn die Variable größer als drei ist. Sonst löscht er das Semaphor (gibt den Besitz der Ressource Zählervariable auf), und kehrt zum Schleifenbeginn zurück. Der Aufruf von DosSleep unterbricht den Thread, und erlaubt OS/2 einem Thread mit derselben oder höheren Priorität, die CPU zu geben. Ohne den Aufruf von DosSleep, würde der Thread in einer immerwährenden Schleife laufen, und unnötig CPU-Zeit verschwenden. Der Tastaturthread braucht nicht DosSleep aufzurufen. Der Parameter IO_WAIT blockiert den Thread bis eine Tastatureingabe ansteht.

Der Tastaturthread benützt ebenso die Semaphore CountSem, um auf die Zählervariable zuzugreifen. Bei jedem Tastendruck auf die Escape-Taste, greift der Tastaturthread auf die Semaphore zu, und blockiert, bis das Semaphor gelöscht ist. Er erhöht dann die Zählervariable und löscht das Semaphor. Dieser Mechanismus vermeidet es, daß auf die Zählervariable in derselben Zeit zugegriffen wird, wie dies der Hauptthread tut. So ist der Zugriff auf die Zählervariable serialisiert, und die beiden Threads sind synchronisiert. Nach diesem einfachen Beispiel können wir uns jetzt ein etwas komplexeres ansehen: eine Multithread-Version unseres HELLO.C

Ein Multithread HELLO.C

Die Multithread-Version von HELLO.C zeigt die Verwendung von Threads und Semaphoren, für den seriellen Zugriff und die Kontrolle der Ressourcen. Das Original HELLO.C zeigt einfach eine Nachricht auf dem Bildschirm an und beendet sich. HELLO0.C (gezeigt im *Bild 14*) teilt den Bildschirm in Bereiche und gibt Nachrichten in diesen Bereichen aus. Das Programm teilt jedem Bereich einen Thread zu, der für das Anzeigen und Löschen dieser Nachricht zuständig ist. Die Threads des Programms schreiben und löschen ihre Nachrichten, bis der Anwender die Escape-

```
/* hello0.c RHS 10/14/88
 *
 * OS/2 and 1988 version of K&R's hello.c
 * demonstrates multiple threads
 */

/*
This program provides an introduction to the use of threads and semaphores
under OS/2.  It divides the screen up into a series of logical frames.
Each frame is a portion of the screen that is managed (written to) by a
single thread.  The exact number of frames will depend on the current
screen length (25, 43 and 50 lines).  Each thread has its own data from
which it knows where the frame can be found on screen.  This includes
a semaphore which signals the thread when to proceed.  These elements can
be found in the FRAME data type.

Upon receiving a signal from its semaphore (i.e., the semphore has been
cleared), the thread either draws a message on the frame or clears the
frame, and reverses the flag that determines this.  Then it again blocks
until its semaphore has been cleared again.

The main program thread starts by setting up the frame information:
checking the screen size, determining the number and size of the frames.
It also "randomly" selects the order in which the frames will appear.

Then it sets each thread's semaphore, and initiates each thread (remember
the threads will block until their semphores are cleared.

Finally, the main program goes into an infinite loop, clearing each thread's
semaphore, sleeping for at least 1 millisecond, and then continuing to the
next thread.  Thus the threads asynchronously call the VIO subsystem to
draw or clear each frame, while the main program thread continues.

An optional parameter can be passed to set the number of milliseconds passed
to DosSleep, allowing the operator to more accurately "see" the order in
which the frames appear/erase.  This value must always be at least 1 to
allow the main program thread to give time to the CPU scheduler.

A call to _beginthread() early in main() sets up a thread to monitor key
board input.  This thread blocks until a key is pressed, then examines the
key, and if they is the Escape Key (27 decimal or 1bH), the thread calls
DosExit to kill the whole process.
 */

#define INCL_SUB
#define INCL_DOS
#include<stdio.h>
#include<string.h>
#include<assert.h>
#include<stdlib.h>
#include<process.h>
#include<os2.h>

#if !defined(TRUE)
#define TRUE    1
#endif

#if !defined(FALSE)
#define FALSE   0
#endif

#define LINES25     4                          /* height in lines of frames*/
#define LINES43     6
#define LINES50     7
#define RANDOMIZER  5
#define MAXFRAMES   28                         /* limited to max frames
possible */
#define RAND()      (rand() % maxframes);
#define THREADSTACK 400                        /* size of stack each thread*/
#define IDCOL       15
#define ESC         0x1b

char *blank_str =                              /* string for blanking frame*/
    "                  ";
    /* frame data ************************/
char *hello_str25[LINES25+1] =
    {
    "┌────────────────┐",
    "│  Hello, world! │",
    "│ from Thread #  │",
    "└────────────────┘",
    "\0"
    };

char *hello_str43[LINES43+1] =
    {
    "┌────────────────┐",
    "│                │",
    "│  Hello, world! │",
    "│ from Thread #  │",
    "│                │",
    "└────────────────┘",
    "\0"
    };

char *hello_str50[LINES50+1] =
    {
    "┌────────────────┐",
    "│                │",
    "│  Hello, world! │",
    "│                │",
    "│ from Thread #  │",
    "│                │",
    "└────────────────┘",
    "\0"
    };


char **helloptr;
int numlines;

typedef struct _frame                          /* frame structure        */
    {
    unsigned    frame_cleared;
    unsigned    row;
    unsigned    col;
    unsigned    threadid;
    long        startsem;
    char        threadstack[THREADSTACK];
    } FRAME;

FRAME far *frames[MAXFRAMES];                  /* pointers to frames     */
unsigned maxframes;
unsigned curframe;
long sleeptime = 1L;                           /* minim sleep time       */
char keythreadstack[THREADSTACK];

    /* function prototypes ***************************/
void hello_thread(FRAME far *frameptr);
void keyboard_thread(void);
void main(int argc, char **argv);
```

Bild 14: (Fortsetzung und Ende)

```
void main(int argc, char **argv)
    {
    int row, col, maxrows, maxcols, len, i, loops = 0;
    VIOMODEINFO viomodeinfo;

    if(argc > 1)
        sleeptime = atol(argv[1]);
    if(sleeptime < 1L)
        sleeptime = 1L;

                                            /* start keyboard thread    */
    if(_beginthread(keyboard_thread,keythreadstack,THREADSTACK,NULL) == -1)
        exit(-1);

    viomodeinfo.cb = sizeof(viomodeinfo);
    VioGetMode(&viomodeinfo,0);             /* get video info           */

    maxrows = viomodeinfo.row;
    maxcols = viomodeinfo.col;

    switch(maxrows)
        {
        case 25:
            helloptr = hello_str25;
            numlines = LINES25;
            break;
        case 43:
            helloptr = hello_str43;
            numlines = LINES43;
            break;
        case 50:
            helloptr = hello_str50;
            numlines = LINES50;
            break;
        default:                            /* fail if not 25,43,50 lines*/
            assert(0);
            exit(-1);
        }

    len = strlen(*helloptr);


    maxframes = (maxrows / numlines) * (maxcols / len);


    assert(maxframes <= MAXFRAMES);

    for( i = 0; i < maxframes; i++)         /* initialize structures    */
        {
        if(!(frames[i] = malloc(sizeof(FRAME))))
            exit(0);
        frames[i]->frame_cleared = FALSE;
        frames[i]->startsem = 0L;
        memset(frames[i]->threadstack,0xff,sizeof(frames[i]->threadstack));
        }

    i = RAND();                             /* get first random frame   */

                                            /* set up random appearance */
    for(row = col = 0; loops < maxframes ; )  /* set row/col each frame   */
        {
        if(!frames[i]->frame_cleared)
            {
            frames[i]->frame_cleared = TRUE;  /* set for empty frame      */
            frames[i]->row = row;           /* frame upper row          */
            frames[i]->col = col;           /* frame left column        */


            col += len;                     /* next column on this row  */

            if(col >= maxcols)              /* go to next row?          */
                {
                col = 0;                    /* reset for start column   */
                row += numlines;            /* set for next row         */
                }

            i = RAND();                     /* get next random frame    */
            }
        else
            ++i;

        if(i >= maxframes)
            {
            i -= maxframes;
            loops++;                        /* keep track of # of frames*/
            }
        }


    for( i = 0 ; i < maxframes; i++)        /* start a thread for each  */
        {
        DosSemSet(&frames[i]->startsem);    /* initially set each sem.  */

                                            /* start each thread        */
        if((frames[i]->threadid = _beginthread(
                (void far *)hello_thread,
                (void far *)frames[i]->threadstack,
                THREADSTACK,
                (void far *)frames[i])) == -1)
            {
            maxframes = i;                  /* reset maxframes on failure*/
            break;
            }
        }


    while(TRUE)                             /* main loop                */
        {
                          /* swing thru frames, signalling to threads */
        for( i = 0; i < maxframes; i++)
            {
            DosSemClear(&frames[i]->startsem);  /* clear: thread can go  */
            DosSleep(sleeptime);            /* sleep a little           */
            }
        }
    }


void hello_thread(FRAME far *frameptr)      /* frame thread function    */
    {
    register char **p;
    register int line;
    int len = strlen(*helloptr);
    unsigned row, col = frameptr->col;
    char idstr[20];


    while(TRUE)
        {
        DosSemRequest(&frameptr->startsem,-1L);   /* block until cleared */
        itoa(frameptr->threadid,idstr,10);        /* init idstr          */
        row = frameptr->row;                      /* reset row           */

        if(!frameptr->frame_cleared)              /* if frame in use, erase*/
            for( line = 0; line < numlines; line++, row++)
                VioWrtCharStr(blank_str,len,row,col,0);
        else                                      /* else frame not in use*/
            {
            p = helloptr;                         /* print message       */
            for( line = 0; **p; line++, row++, p++)
                VioWrtCharStr(*p,len,row,col,0);
                                                  /* write id # in frame */
            VioWrtCharStr(idstr,3,row-(numlines/2),IDCOL+col,0);
            }
        frameptr->frame_cleared = !frameptr->frame_cleared;/*toggle use flag
*/
        }
    }

void keyboard_thread(void)
    {
    KBDKEYINFO keyinfo;

    while(TRUE)
        {
        KbdCharIn(&keyinfo,IO_WAIT,0);      /* wait for keystroke       */
        if(keyinfo.chChar == ESC)           /* if ESC pressed, break    */
            break;
        }
    DosExit(EXIT_PROCESS,0);                /* terminate the process    */
    }

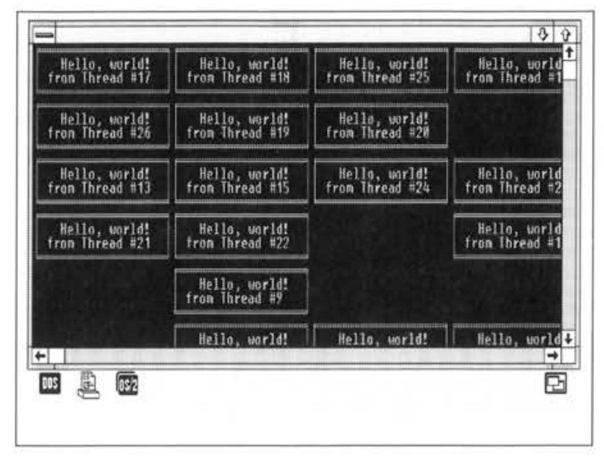
    /******** end of hello0.c *********/
```

Taste benutzt. Dieses bereichbasierende Format ist die Basis für andere Beispielprogramme, während wir die OS/2-Untersysteme und andere Fähigkeiten in späteren Artikeln untersuchen.

Jeder Thread eines Bereichs erhält einen Zeiger auf die Datenstruktur des Bereichs, der die Informationen enthält, die der Thread benötigt. Die Struktur beinhaltet die Zeilen-/Spalten-Koordinaten des Bereichs, die ID des Threads (die von _beginthread zurückgegeben wird), ein Semaphore das der Hauptthread benötigt, um den Thread zu aktivieren, und den Stack des Threads. Die Anzahl der Bereiche, die auf dem Bildschirm erscheinen, hängt vom Videomodus zur Laufzeit des Programms ab (25, 43 oder 50 Zeilen).

HELLO0.C kann ein zusätzliches Kommandozeilenargument übergeben werden, die Anzahl der Millisekunden, die der Hauptthread zwischen den Aktivierungen der Bereichs-Threads schlafen soll. Dies gestattet Ihnen, den Vorgang zu verzögern, so daß Sie gut erkennen können, was vor sich geht. Der Kommandozeilenparameter wird in der Variablen sleep_time gespeichert und ist mit einer Millisekunde vorbelegt, welche die DosSleep-Kernfunktion auf 32 Millisekunden aufrundet. Diese Zeit von 32 Millisekunden ist die minimale OS/2-Zeitscheibe. Dies verhindert auch, daß HELLO0.C zu viel CPU-Zeit vergeudet. Sie können das Programm auch mit den Parametern 50, 100, 500 oder 1000 (gleichbedeutend mit einer Sekunde) laufen lassen, um besser zu sehen, was vor sich geht.

Das Programm prüft zu Beginn den Kommandozeilenparameter und ruft dann _beginthread auf, um einen Tastaturthread zu starten, der blockiert, bis eine Tasteneingabe durchgeführt wird, und dann das Programm beendet, wenn der Benutzer die Escape-Taste betätigt. Dieser Programmteil stammt direkt vom vorhergehenden Beispiel, das Sie im Bild 8 sehen. Als nächstes erledigt der Hauptthread einige Verwaltungsarbeiten: er holt sich den Videomodus durch den Funktionsaufruf VioGetMode, initialisiert die Anzahl der Bildschirmzeilen und be-



► *Bild 15: Jeder Bereich von Hello, der hier im Textfenster des Presentation Managers läuft, wird von einem eigenen Thread erzeugt, der durch OS/2 RAM-Semaphore kontrolliert wird.*

rechnet die maximale Anzahl der Bereiche, die er anzeigen soll. Dann allokiert und initialisiert er die Bereichsstrukturen (Datentypen FRAME), wählt zufällig einen FRAME für jeden Bildschirmbereich und weist die zugehörigen Zeilen-/Spalten-Koordinaten zu. Deshalb scheinen die Bereiche in einer zufälligen Reihenfolge am Bildschirm zu erscheinen und zu verschwinden, obwohl diese Ordnung während des Programmlaufs dieselbe ist.

Der wirkliche Spaß in HELLO0.C beginnt, wenn der Hauptthread DosSemSet für jeden FRAME aufruft, auf den dann der Aufruf _beginthread folgt, um den FRAME-Thread zu starten (der blockiert, bis das Semaphor gelöscht wird). Der Hauptthread bearbeitet dann bis zum Programmende eine Schleife. In dieser Schleife wird der Thread für jeden FRAME durch das Löschen des FRAME-Semaphors aktiviert. Der Aufruf von DosSleep unterbricht den Thread, gibt die Zeitscheibe auf, bevor der nächste Thread aktiviert wird. Der Hauptthread erledigt dies für jeden Bereich und wiederholt den Vorgang anschließend. Der Aufruf von DosSleep in einer Schleife gestaltet das Programm effizienter, da keine CPU-Zeit verschwendet wird.

Was erledigt jeder FRAME-Thread? Der Programmteil jedes FRAME-Thread ist in der Funktion hello_thread enthalten, die einen Parameter benötigt, die Adresse des FRAME des Threads, die ihm von _beginthread übergeben wird. Der Code des Hauptthread besteht aus einer Schleife, an deren Spitze der Aufruf der Kernfunktion DosSemRequest steht. Durch die Übergabe der Adresse des Semaphors und einer -1 an diese Funktion blockiert der Thread, bis das Semaphor gelöscht wird. So ist jeder Thread untätig, bis der Hauptthread das Semaphor löscht.

Nach der Aktivierung eines Threads löscht oder schreibt er seine Meldung, abhängig von einer Variablen, die bei jedem Aufruf geändert wird. Die Funktion VioWrtCharStr des Vio-Untersystems erledigt die komplette Bildschirmausgabe, indem sie eine bestimmte Anzahl von Zeichen eines Strings an eine bestimmte Position des Bildschirms ausgibt. Ein Beispiel der Ausgabe eines FRAME-Threads sehen Sie im *Bild 15.*

Das Programm läuft, bis der Anwender die Escape-Taste betätigt. Hierdurch wird der Tastaturthread aufgeweckt (ohne Tastatureingabe war er blockiert) und beendet das Programm.

HELLO0.C ist wahrscheinlich Multithread-Overkill (wie oft benötigt eine Applikation 25 Threads?), aber es verdeutlicht die Multithread-Betrachtungen in diesem Artikel. Mit diesen Grundlagen, können Sie komplexere Programme schreiben, die mehrere Threads besitzen. Tatsächlich gestaltet sich die Programmentwicklung noch interessanter, wenn wir in der nächsten Ausgabe das VIO-Untersystem erkunden.

Richard Hale Shaw

Bild 11: DosExitList

DosExitList

OS/2 erlaubt einem Prozeß, eine Anzahl von Routinen einzurichten, die beim Programmende immer aufgerufen werden. Typischerweise sind dies Funktionen, die die Ressourcen des Prozesses freigeben (wie das Schließen offener Dateien). Unabhängig wann und wie der Prozeß sich beendet, OS/2 ruft diese Funktionen am Ende auf. Eine Applikation kann Funktionen, die OS/2 aufruft, registrieren und so gewährleisten, daß ein korrektes Ende und die Freigabe der Ressourcen des Prozesses durchgeführt wird.

Die Kernfunktion DosExitList registriert die Funktionen und beendet die Funktionen auch. Der Funktionsprototyp ist:

```
unsigned DosExitList(unsigned code, void far
*fptr(unsigned));
```

Beim Registrieren einer Funktion kann der erste Parameter entweder EXLST_ADD oder EXLST_REMOVE sein, der die Funktion in die Liste einfügt oder aus der Liste löscht. Durch das dynamische Einfügen und Löschen in der Liste, kann der Prozess die Hinterlassenschaft nach seinem Tode regeln (in einer Art, die dem Menschen ähnlich ist). Ein Prozeß kann eine Funktion vor dem Ende aus der Liste löschen, wenn sie nicht länger benötigt wird. Der zweite Parameter ist ein Zeiger auf die Funktion, die registriert werden soll.

Wenn OS/2 die Ausführung der Endefunktionen startet, sind der Prozeß und alle Threads zerstört, nur der Thread, der die DosExitList Funktionen ausführt lebt noch. OS/2 übergibt die Kontrolle an jede eingetragene Funktion, aber in keiner bestimmten Reihenfolge. Nach der Ausführung von allen registrierten Funktionen beendet OS/2 den Prozeß.

Die Endefunktionen, die mit DosExitList eingetragen sind, müssen im Prozeß vorhanden sein (also in seinem Code Segment) und sollten so kurz und sicher wie möglich sein. Eine Endefunktion darf alle OS/2-Systemaufrufe durchführen, außer DosCreateThread und DosExecPgm.

Eine grobe Definition einer Endefunktion sehen Sie im folgenden:

```
void far termfunc(unsigned code)
{
        if(code != TC_EXIT)
        {
                /* Ende Behandlung */
        }
        DosExitList(EXLST_EXIT, 0);
}
```

Eine Endefunktion verfügt über einen Parameter und keinen Rückgabewert. Der Parameter kann einen der folgenden Bedeutungen haben:

TC_EXIT	Normales Ende
TC_HARDERROR	Ende durch harten Fehler
TC_TRAP	Trap Operation
TC_KILLPROCESS	Nicht abgefangener DosKillProcess

Dies gestattet der Endefunktion, zu entscheiden, ob ein normales Ende des Prozesses vorliegt. Sie kann auch entscheiden, welche Aktionen vorgenommen werden sollen.

Endefunktionen müssen sich selbst durch den Aufruf DosExitList(EXLST_EXIT, 0) beenden. Sie können kein return ausführen (explizit oder implizit, durch die schließende geschweifte Klammer der Funktion), da sonst der Prozeß hängt, und nie beendet wird. Der Code EXLST_EXIT sagt OS/2, daß die Endebehandlung fertig ist, und die nächste Funktion der Liste aufgerufen werden kann.

Einsatz der Multithread-Bibliothek: DosCreateThread oder _beginthread

Das OS/2 API-Interface gestattet das Erzeugen und Beenden von Threads durch die Funktionen DosCreateThread und DosExit. Obwohl der Code des Threads in einer Funktion enthalten sein muß, können Sie dieser keine Parameter übergeben, wenn Sie DosCreateThread verwenden. Wenn Sie mehr eine C-ähnliche Schnittstelle für ein Multithread-Programm bevorzugen, oder die Multithread-Standardbibliothek LLIBCMT.LIB benutzen wollen, so sollten Sie sich mit einer anderen Schnittstelle (_beginthread und _endthread) vertraut machen. Wenn Sie in Ihren Threads Standardbibliotheks-Funktionen verwenden wollen, so ist der Gebrauch von LLIBCMT.LIB ein Muß.

Der Fall printf

Die Diskussion über den Gebrauch von printf im Text, verdeutlicht diese Notwendigkeit. Eine Funktion wie printf benötigt einen großen internen Puffer für die Formatierung des Ausgabestrings. Obwohl dieser Puffer genügt, wenn ein einzelner Thread diese Funktion ausführt, ist die Ausgabe undefiniert, sobald mehr als ein Thread printf zur selben Zeit ausführt. Es gibt zwei Wege, wie printf geschrieben werden kann, um dies zu lösen. Sie können ein Semaphor verwenden, um nur einem Thread den Zugriff auf printf zu gewähren, oder Sie können einige Semaphore verwenden, um einer begrenzten Anzahl von Threads den gleichzeitigen Zugriff auf printf zu ermöglichen.

Sehen wir uns diese zwei Lösungen näher an. Bei der ersten Methode (in *Listing A* skizziert) wird am Beginn von printf ein Semaphor gesetzt und am Ende gelöscht. Diese Lösung serialisiert den Programmteil von printf, so daß nur ein Thread jeweils die Funktion ausführen kann. Dies bedeutet aber auch, daß bei jedem Aufruf von printf der Thread blockiert wird, wenn ein anderer Thread die Funktion printf aufgerufen hat, und blockiert bleibt, bis der andere Thread das Semaphor löscht. Es gibt keine Garantie, daß der nächste Thread printf ausführen kann. OS/2 kann nicht gewährleisten, daß der nächste Thread, der printf aufrufen will, dies auch kann. So kann ein Thread mit niedrigerer Priorität ständig von einem Thread mit höherer Priorität verdrängt werden, der printf ausführt – was zu unerwünschten optischen Ergebnissen führen kann.

Die zweite Methode beschränkt die Anzahl der Threads, die gleichzeitig die Funktion printf ausführen können. Sie ergibt aber auch keine Kollisionen zwischen Threads, die sich um den Zugriff auf printf bewerben. Bei dieser Lösung (in *Listing B* zu sehen) verwendet printf eine feste Anzahl

```
void printf(char *fmt,...)
    {
static long printfSem = OL;
static char formatbuffer[BUFSIZ];

DosSemRequest(&printfSem,-1L);

•
•
•

DosSemClear(&printfSem);
}
```

◂ *Listing A: Eine Skizze der Version 1 der Multithread-Funktion printf.*

```
#define    MAXTHREADS    32
void printf(char *fmt,...)
{
static long printfSems[MAXTHREADS] =
    {OL,OL,OL,OL,OL,OL,OL,OL,OL,OL,
     OL,OL,OL,OL,OL,OL,OL,OL,OL,OL,
     OL,OL,OL,OL,OL,OL,OL,OL,OL,OL,
     OL,OL};

char formatbuffers[MAXTHREADS][BUFSIZ];
int semno;

for(semno = 0; semno < MAXTHREADS; semno++)
    if(!DosSemRequest(&printfSems[semno],OL))
        break;
assert(semno < MAXTHREADS);

•
•
•

DosSemClear(&printfSem[semno]);

}
```

◂ *Listing B: Eine Skizze der Version 2 der Multithread-Funktion printf.*

```
#include<mt\process.h>
#include<mt\stddef.h>

int cdecl far _beginthread(
         void (cdecl far *start_address) (void far *),
         void far *stack_end,
         unsigned stack_size,
         void far *arglist);

•
•
•
void far cdecl _endthread(void)
```

◂ *Listing C: Funktionsprototypen von _beginthread und _endthread.*

von Formatierungspuffern, deren Zugriffe durch Semaphore kontrolliert werden. Daher erhält jeder Thread während der Ausführung der Funktion seinen eigenen Puffer. Der Nachteil ist die Beschränkung auf eine Anzahl von Puffern, wodurch nur eine bestimmte Anzahl von Threads gleichzeitig zugreifen kann.

Aus diesem Grund ist _beginthread vorhanden. Diese Funktion bietet ein C-ähnliches Interface zum Erzeugen eines neuen Threads, indem Sie dem Thread Parameter übergeben dürfen, und bei erfolgreicher Erzeugung die Thread-ID zurückbekommen. Hier ist aber der aufrufende Prozeß auf 32 Threads beschränkt, weit weniger als die 255 Threads, die mit DosCreateThread erzeugt werden können. Für die meisten Applikationen sind aber 32 Threads mehr als genug. Hier kann die Multithread-Bibliothek LLIBCMT.LIB zum Einsatz gebracht werden, die davon ausgeht, daß nicht mehr als 32 Threads je Prozeß vorhanden sind. Aus diesem Grund ist die Funktion _beginthread nur vorhanden, wenn Sie diese Bibliothek verwenden, und Sie sollten diese auch statt DosCreateThread benutzen, wenn Sie die Bibliothek verwenden.

Einsatz von _beginthread

Wie arbeitet _beginthread? Aus dem Funktionsprototyp in *Listing C* sehen Sie, daß diese Funktion wie DosCreateThread auch die Adresse der Funktion benötigt, die den Programmteil des Threads enthält. Im Gegensatz zu DosCreateThread benötigt _beginthread aber nicht die Adresse des Endes des Stacks, sondern die Adresse des Stacks, so wie Sie ihn in C deklarieren (also den Beginn des Stacks). Daher müssen Sie auch die Größe des Stacks angeben (im dritten Parameter). Zu guter Letzt akzeptiert _beginthread noch einen Parameter, der die Funktion einfacher gestaltet als DosCreateThread. Einen Argumentparameter für die Übergabe von Argumenten an die Threadfunktion. Ein Beispiel dafür sehen Sie in dem Multithread-Programm HELLO0.C. Die Funktion _beginthread liefert -1 im Fehlerfalle, oder die Thread-ID des neuen Threads.

Selbstverständlich muß _beginthread irgendwo DosCreateThread aufrufen. Die Funktion ruft sogar DosCreateThread auf, wenn Sie die Anzahl der erlaubten Threads der Multithread-Bibliothek überschritten haben. Die einzige Möglichkeit, die sie besitzt, ist die Rückgabe der Thread-ID -1, falls die zurückgegebene Thread-ID von DosCreateThread größer als 32 ist. Dies setzt aber zwei undokumentierte OS/2-Eigenschaften voraus. Erstens, daß DosCreateThread immer die niedrigste verfügbare Thread-ID zurückgibt, und daß OS/2 die ID von beendeten Threads wieder verwendet.

Linken von LLIBCMT.LIB

Der Einsatz von _beginthread und dem Äquivalent von DosExit, _endthread (auch in Bild C gezeigt) setzt voraus, daß die Bibliotheken LLIBCMT.LIB und DOSCALLS.LIB im aktuellen, oder in dem Verzeichnis sind, auf das die LIB-Variable (im Environment, oder in der MAKE-Datei – sehen Sie auch die MAKE-Datei von HELLO0.C) zeigt. Die Bibliothek DOSCALLS.LIB wird benötigt, da _beginthread, _endthread und einige andere Bibliotheksfunktionen OS/2 API-Funktionen aufrufen. LLIBCMT.LIB sollte statt den anderen Laufzeitbibliotheken verwendet werden.

Zusätzlich sollten Sie die INCLUDE-Variable (auch im Environment, oder in der MAKE-Datei) ändern, so daß sie auf das Verzeichnis MT zeigt, das der Compiler unter dem Standard INCLUDE-Verzeichnis einrichtet. Dieses Verzeichnis enthält Kopien der Standard-Headerdateien, und sollte für Programme mit mehreren Threads verwendet werden. Sie können die INCLUDE-Variable auch mit der Compiler-Kommandozeilenoption /I setzen. Die Funktionsprototypen für _beginthread und _endthread finden Sie in der Headerdatei PROZESS.H.

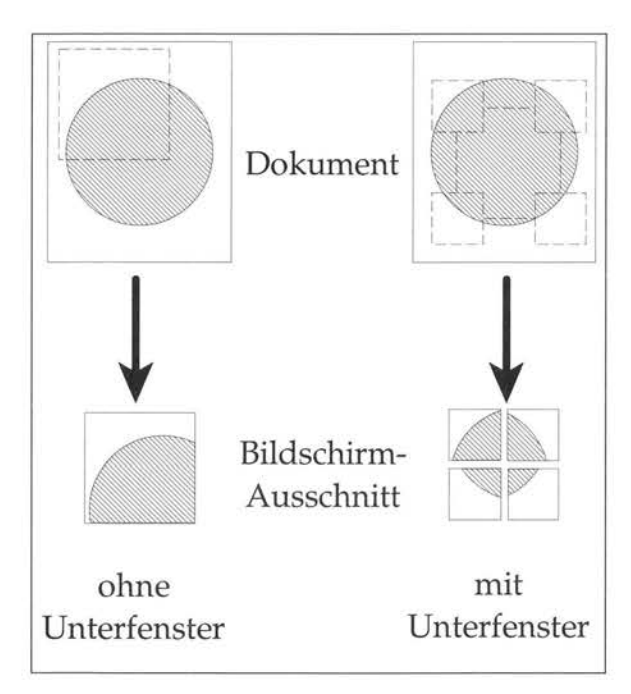
Beschränkungen

Multithread-Programme müssen bei der Ausführung mehrere Annahmen machen. Alle Programm- und Datenadressen müssen far sein. Zusätzlich muß das Programm annehmen, daß das Datensegment fest ist, sollte aber nicht davon ausgehen, daß das Stack- und das Datensegment gleich sind. Die Stacküberprüfung zur Laufzeit muß abgeschaltet sein. Sie können Compiler Kommandozeilen-Optionen verwenden, die diese Aufgaben erledigen. Dies sind /Alfw und /G2s. Ebenso sollte die Compiler-Option /Zl oder die Linker-Option /NOD verwendet werden, damit nicht die Standardbibliothek gelinkt wird. Die MAKE-Datei von HELLO0.C verdeutlicht dies, und kann leicht für das Übersetzen und Linken Ihrer Multithread-Applikation geändert werden.

Selbstverständlich können Multithread-Programme nicht zu kombinierten Applikationen umgewandelt werden, da MS-DOS keine Möglichkeiten der gleichzeitigen Ausführung von mehreren Threads bietet.

Unterfenster und entfernbare Speicherobjekte sinnvoll angewandt:

Eine Hex-Dump-Applikation für Windows



◂◂ *Bild 1: Prinzip der Unterfenster-Darstellung.*

Programme zum Anzeigen von Dateien in der HEX-Darstellung gibt es zahlreich – nicht nur unter MS-DOS, sondern inzwischen auch unter MS-Windows. Die im folgenden beschriebene Windows-Applikation unterscheidet sich jedoch von den üblichen durch den Einbau von Unterfenstern und der impliziten Verwendung der virtuellen Speicherverwaltung von Windows. In dem Artikel werden auch einige Hinweise gegeben, wie man schnelle Textausgabe unter Windows realisieren kann.

Das nächste, was der Benutzer bei Windows nach den bunten Grafiken bemerkt, ist die Möglichkeit, mehrere Programme gleichzeitig anzeigen zu können – etwas, was er bei DOS lange vermißt hat. Diese Fähigkeit erlaubt natürlich auch die Anzeige ein- und desselben Dokuments durch das Laden in mehreren Applikationen gleichzeitig. Diese Vorgehensweise ist aber wenig sinnvoll: Änderungen an dem Dokument, die in der einen Applikation gemacht werden, wirken sich nicht automatisch auf die Dokumente in den anderen Applikationen aus. Dies kann sich beim Abspeichern verhängnisvoll auswirken – sofern es das Betriebssystem überhaupt erlaubt, ein- und dieselbe Datei mehrfach aufzurufen und zu verändern. Auf jeden Fall werden Veränderungen an dem Dokument, die versehentlich in verschiedenen Applikationen gemacht werden, nicht korrekt abgespeichert.

Nun ist es aber oft wichtig, von einem Dokument mehrere Ansichten gleichzeitig am Bildschirm zu haben. Aus diesem Grund hat Microsoft die MDI-Schnittstelle eingeführt [2] (MDI=*Multiple Document Interface*, zu deutsch Mehrfach-Dokumentenschnittstelle). Sie erlaubt die gleichzeitige Anzeige von mehreren Dokumenten in separaten Fenstern innerhalb einer Applikation. Wird das gleiche Dokument in mehreren Fenstern angezeigt, werden die einzelnen Kopien des Dokuments in den Kopfzeilen der Fenster durchnumeriert und Veränderungen, die in einem Fenster gemacht werden, automatisch auf die anderen Fenster übertragen und angezeigt, sofern die gleichen Ausschnitte abgebildet werden. Alle Veränderungen betreffen jedoch immer nur *ein* Dokument, das lediglich mehrfach angezeigt wird. Eine ordnungsgemäße und konsistente Bearbeitung ist daher sichergestellt. Die MDI-Schnittstelle dient aber in erster Linie dazu, entweder verschiedene Dokumente gleichzeitig anzuzeigen oder ein einzelnes Dokument in völlig unterschiedlichen Darstellungen zu präsentieren. Bei MS-Excel beispielsweise kann ein Dokument wahlweise als Tabelle oder als Diagramm angezeigt werden. Für die mehrfache Abbildung eines Dokuments bietet MS-Excel eine einfachere Technik, die der Unterfenster (im Original *panes*).

Unterfenster und ihre Anwendung

Pane heißt wörtlich übersetzt »Festerscheibe«, eigentlich ein sehr passender Ausdruck, besteht doch ein altmodisches Sprossenfenster aus mehreren einzelnen Scheiben, die nebeneinanderliegen. Zum Verwechseln ähnlich sehen Unterfenster aus, wenn man vom abstrakten Fensterbegriff von Windows ausgeht. Da die Übersetzer der MS-Excel-Dokumentation aber *pane* mit »Un-

► *Bild 2: Unterfenster-Darstellung beim MS-Excel.*

►► *Bild 3: Die Applikation DUMP ohne Unterfenster.*

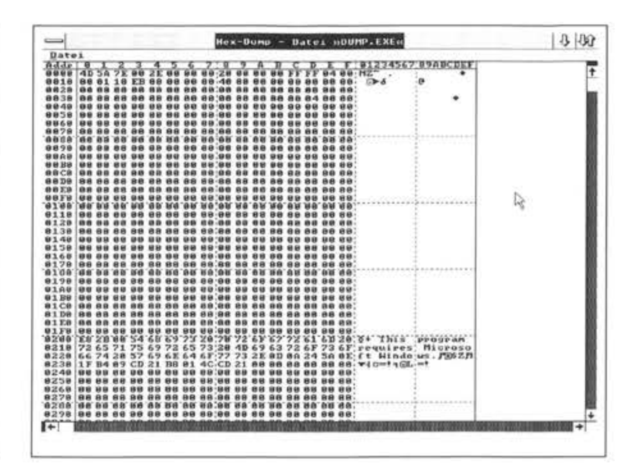
Microsoft Excel - PLATTEN.XLS

Datei Bearbeiten Formel Format Daten Optionen Makro Fenster ?

A1 Interpret

	A		B	C	D
1	Interpret	Titel	Titel	Typ	Anz.
2	Beethoven / Remoortel	Sinfonie Nr. 1	Sinfonie Nr. 1	KLA	1
3	Beethoven / Remoortel	Sinfonie Nr. 7	Sinfonie Nr. 7	KLA	1
4	Debussy / Previn	Images / Faun	Images / Faune	KLA	1
5	Debussy / Ravel / Abbado	Nocturnes / D	Nocturnes / Daphnis	KLA	1
6	Haendel / Rogg / Armand	16 Konzerte f. (	16 Konzerte f. Orgel & Orchester	KLA	1
7	Mahler / Kubelik	Symphonie Nr	Symphonie Nr. 1	KLA	1
8	Ravel / Ansermet	Daphnis / Vals	Daphnis / Valse / Pavane	KLA	1
9	Ravel / Bernstein	Klavierkonzert	Klavierkonzert G-Dur	KLA	1
10	Ravel / Debussy / Solti	Bolero / La Me	Bolero / La Mer / Faune	KLA	1
11	Ravel / Horne / Bernstein	Sheherazade	Sheherazade	KLA	1
12	Ravel / Ozawa	Ma Mere l'Oye	Ma Mere l'Oye	KLA	1
13	Ravel / Skrowaczewski	Daphnis / Ma	Daphnis / Ma Mere l'Oye	KLA	1
14	Diverse	Charisma Dist	Charisma Disturbance	POP	2
15	Diverse	German Rock	German Rock Scene	POP	1
16	Diverse	German Rock	German Rock Scene Vol. II	POP	1
17	Diverse	German Supe	German Super Rock	POP	1
18	Diverse	Harvest Hits th	Harvest Hits the Road	POP	1
19	Diverse	Kraut Rock	Kraut Rock	POP	3
20	Diverse	Rockwork	Rockwork	POP	2
21	Diverse	Super-Rock-F	Super-Rock-Festival	POP	1
22	Anderson, Jon	animation	animation	POP	1

Bereit



terfenster« übersetzt haben, wollen wir diesen Begriff beibehalten und nur ab und zu auf den Originalbegriff zurückgreifen.

Oft sind Dokumente viel zu groß, als daß sie komplett in einem Fenster angezeigt werden können. Es wird dann nur ein entsprechender Ausschnitt angezeigt (*Bild 1* links), den man mit den Rolleisten verschieben kann. Dieser Ausschnitt ist gewöhnlich zusammenhängend. Oft möchte man sich aber unzusammenhängende Teile gleichzeitig ansehen, auch wenn die einzelnen Teile dann kleiner sind. Hierzu dienen Unterfenster (*Bild 1* rechts). In allen Unterfenstern können beliebig Bearbeitungen am Dokument durchgeführt werden, die sich sofort auf die anderen Unterfenster übertragen – denn wie bei MDI existiert weiterhin nur ein Dokument, das lediglich mehrfach angezeigt wird.

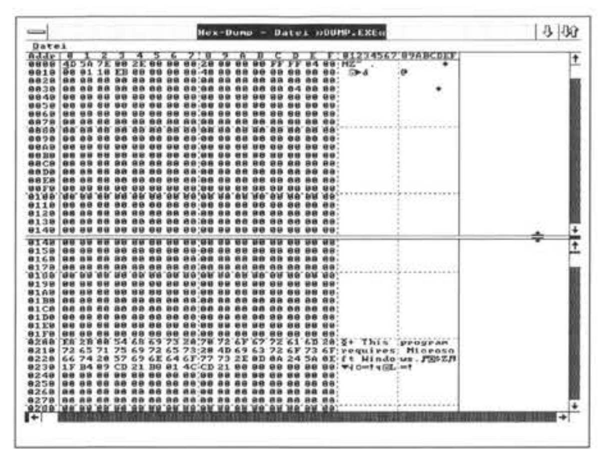
Sie werden sich vielleicht fragen, ob denn Unterfenster wirklich notwendig sind und nicht nur eine Spielerei darstellen. Bei MS-Excel (*Bild 2* zeigt es mit vier Unterfenster) mögen sie aufgrund der Tabellenform ja recht praktisch sein. Aber sonst? Die Anwort darauf lautet, daß Unterfenster immer dann nützlich sind, wenn Dokumente bearbeitet werden müssen, die nicht komplett in einem Fenster dargestellt werden können. Rolleisten und Unterfenster gehören also zusammen. Für die Zweifler sollen im folgenden noch einige Beispiele aus der Praxis gezeigt werden.

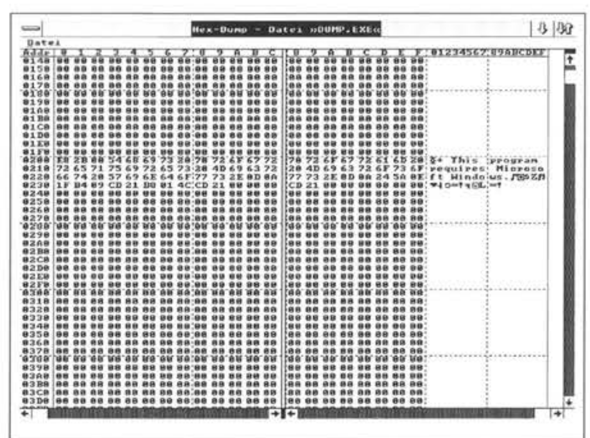
• Sie wollen in einem längeren Text ein Kapitel mit den verwendeten Abbildungen zusammenstellen. Hierzu müssen Sie die Bildunterschriften, die im Text verstreut sind, »einsammeln« und in eine Tabelle kopieren. Das Suchen der Unterschriften kann mit dem Suchbefehl des Textverarbeitungsprogramms erfolgen, aber da die Zwischenablage nur eine Bildunterschrift aufnehmen kann, müssen Sie ständig im Dokument zwischen dem Kapitel mit der Tabelle und dem nächsten Bild hin- und herspringen, was sehr zeitaufwendig ist. Wenn das Textverarbeitungsprogramm dagegen Unterfenster hat, können Sie aus dem Dokument zwei machen, wobei im ersten Unterfenster das Kapitel mit der Tabelle steht und im zweiten die aktuelle Suchposition (die nächste Bildunterschrift). Die Übertragung des Bildtextes erfolgt einfach über die Zwischenablage durch Wechsel von einem Unterfenster zum anderen. Der Arbeitsvorgang ist sehr systematisch. Wildes Herumspringen im Text ist überflüssig.

• In einer größeren Zeichnung wollen Sie mit einem Grafik-Programm ein neues Objekt in der rechten unteren Ecke des Dokuments zeichnen, wobei dieses Objekt gewisse Ähnlichkeiten mit einem anderen Objekt in der linken oberen Ecke besitzt. Beide Ecken lassen sich aber nicht mehr in einem akzeptablen Maßstab gleichzeitig am Bildschirm darstellen. Die Objekte unterscheiden sich auch so stark, daß Kopieren nicht der gewünschte Effekt bringt. Ohne Unterfenster bietet sich lediglich die Möglichkeit an, das Dokument vorher auf Papier zu bringen und von dort aus nachzuzeichnen. Mit Unterfenster können Sie dagegen beiden Ecken in getrennten Unterfenstern gleichzeitig darstellen und leicht auch einige Teile des einen Objekts über die Zwischenablage in die andere Ecke bringen, ohne daß sie den Überblick über ihre Zeichnung verlieren.

• In einer Datenbank ist die Kundendatei als Tabelle abgebildet. In jeder Zeile steht ein Kunde. Zunächst kommen Namen, Anschrift, Telefonnummer und weitere Daten, abschließend das Datum und den Auftragswert der letzten Bestellung. Unglücklicherweise passen die Daten horizontal nicht auf den Bildschirm, sodaß sie zwischen Namen und Bestelldaten ständig hin- und her rollen müssen. Wenn Sie sich einen Überblick verschaffen wollen, welche Kunden in letzter Zeit für Sie interessante Kunden waren, wird dies sehr umständlich. Vielleicht besitzt ihr Datenbank-Programm Befehle zum Ausblenden nicht benötigter Spalten, doch diese Befehle müssen Sie erst einmal kennen und zu bedienen wissen. Unterfenster sind dagegen einheitlich zu bedienen: Sie teilen die Tabelle mit einer vertikalen Linie in zwei Unterfenster auf: Links mit dem Namen des Kunden und rechts mit den Bestelldaten. Und schon ist der Überblick hergestellt.

Sie sehen also: Unterfenster sind eine sehr sinnvolle Anwendung mit vielfältigen Nutzungsformen in der Praxis. Unterfenster sind aber nur dann für den Anwender nützlich, wenn sie ohne großen Aufwand angelegt und in ihrer Größe





◄◄ *Bild 4: DUMP mit horizontaler Fenster-Unterteilung.*

◄ *Bild 5: DUMP mit vertikaler Fenster-Unterteilung.*

verändert werden können. MS-WORD besitzt beispielsweise auch Unterfenster in Form seiner Ausschnitte. Aber wieviel Anwender können diese etwa mit der Maus flüssig bedienen? Man weiß eigentlich nie, mit welcher Maustaste welches Eck oder welche Linie im Ausschnittrahmen angeklickt werden muß. Die Windows-Entwickler haben daraus gelernt. Unterfenster bei MS-Windows sind wirklich sehr einfach anzuwenden. Dazu mehr im nächsten Kapitel.

Die Benutzerschnittstelle von Unterfenstern

Das Schöne an Windows für den Benutzer ist, daß sich die Entwickler der Oberfläche tiefgreifende Gedanken gemacht haben, wie man Applikationen nicht nur leicht erlernen, sondern auch vielseitig und schnell bedienen kann. Dies beginnt beim Verschieben von Fenstern, setzt sich beim Bedienen von Dialogelementen fort und betrifft natürlich auch zukünftige Standard-Elemente wie die Unterfenster. Der Entwickler eines Pane-Managers hat natürlich viel Aufwand, alle Vorschläge zu realisieren. Aber wehe, Sie vergessen eine der Funktionen, die bei anderen Applikationen selbstverständlich vorhanden sind: Der Benutzer wird sich ständig ärgern, daß gerade Ihr Programm diese Funktion nicht hat, Sie werden für dieses Mißgeschick beschimpft und plötzlich ist Ihr liebevoll entwickeltes Programm nicht mehr das, was es vorher war! Aus dem Grund wird in diesem Abschnitt alles zusammengetragen, was ich über die Benutzung von Unterfenstern finden konnte. Im wesentlichen beruht dies auf der MS-Excel-Realisierung einerseits und auf der letzten CUA- (*C*ommon *U*ser *A*ccess)-Spezifikation von IBM andererseits, die über die Vorschläge im Style-Guide des SDK [1] hinausgehen. Auch die MS-Excel-Implementierung besitzt nicht alle Vorschläge aus der CUA-Beschreibung, aber man kann davon ausgehen, daß bei der Liebe und Mühe, die die Entwickler von Excel in ihr Produkt gesteckt haben, dies mit dem nächsten größeren Update der Fall sein wird.

Wenn Sie eine Applikation mit Unterfenstern aufrufen, sieht sie nach dem Laden eines Dokuments fast genauso aus wie eine normale Applikation. Am rechten und unteren Fensterrand befinden sich die Rolleisten. Lediglich links von der horizontalen und oberhalb der vertikalen Rolleiste befindet sich ein unscheinbares schwarzes Kästchen (*Bild 3*). Wenn Sie mit der Maus über das Kästchen oberhalb der vertikalen Rolleiste fahren, stellen Sie fest, daß der Cursor eine andere Gestalt annimmt, die dazu animiert, das Kästchen vertikal zu verschieben. Das erreichen Sie dadurch, daß Sie die linke Maustaste drücken und als Ergebnis erscheint eine graue horizontale Linie (»Sprosse«). Diese können Sie jetzt mit gedrückter Maustaste nach unten ziehen. Sobald Sie die Maustaste loslassen, haben Sie zwei übereinanderliegende Unterfenster erzeugt (*Bild 4*). Entsprechend können Sie durch Anklicken des »Bildschirmteilers« neben der horizontalen Rolleiste durch Ziehen einer vertikalen Linie zwei nebeneinanderliegende Unterfenster erzeugen (*Bild 5*). Sie können die Aufteilung in Zukunft beliebig verschieben, indem Sie einfach erneut den Bildschirmteiler anklicken und ihn an die gewünschte Position bewegen. Verschieben Sie Ihn an den Bildschirmrand oder verkleinern Sie ein Unterfenster so weit, daß seine Rolleisten sehr merkwürdig aussehen würden oder die Darstellung im Unterfenster unsinnig wäre, wird das betreffende Unterfenster ganz entfernt und der Bildschirmteiler wieder auf die Ausgangsposition gestellt. Soweit die Unterfenster-Schnittstelle, wie sie auch bei MS-Excel realisiert wurde.

CUA gibt nun noch weitere Vorschläge, die sich in der Praxis als sehr nützlich erweisen. So kann mit der Maus nicht nur der Bildschirmteiler, sondern die gesamte horizontale oder vertikale Trennlinie angeklickt und verschoben werden. Sind beide Trennlinien angezeigt (also ein »Fensterkreuz« sichtbar) kann dieses mit einer Mausbewegung vertikal und horizontal verschoben werden, indem die Mitte des Kreuzes angeklickt wird. Befindet sich ein Bildschirmteiler in Ausgangsposition und wird er doppelt angeklickt, werden er und seine Trennlinie in die entsprechende horizontale oder vertikale Mitte des Dokuments gesetzt (schnelles Halbieren).

▸ Tabelle 1: Tasten-Codes zum Verschieben des Fensterkreuzes nach dem Befehl »Teilen«.

▸▸ Bild 6: Darstellung von vier Unterfenstern.

[←], [↑], [→], [↓] verschieben das Fensterkreuz in die angegebene Richtung um jeweils eine horizontale oder vertikale Einheit.

[Strg]+[←], [↑], [→], [↓] verschieben das Fensterkreuz in die angegebene Richtung um einen Punkt.

[Bild ↑] setzt die horizontale Linie des Fensterkreuzes an den oberen Fensterrand.

[Strg]+[Bild ↑] setzt die vertikale Linie des Fensterkreuzes an den linken Fensterrand.

[Bild ↓] setzt die horizontale Linie des Fensterkreuzes an den unteren Fensterrand.

[Strg]+[Bild ↓] setzt die vertikale Linie des Fensterkreuzes an den rechten Fensterrand.

[Pos 1] setzt das Fensterkreuz in die linke obere Ecke des Hauptfensters.

[Ende] setzt das Fensterkreuz in die rechte untere Ecke des Hauptfensters.

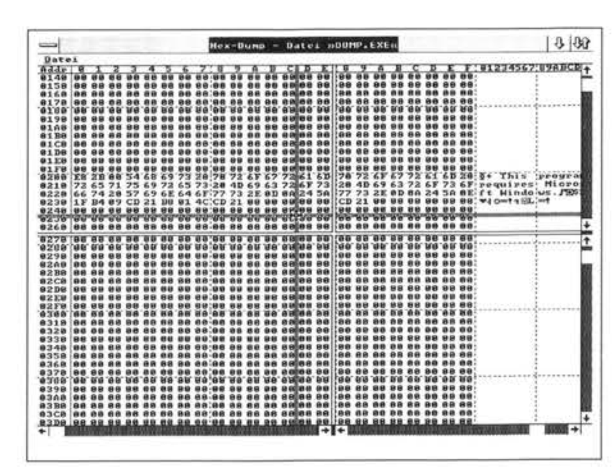
[Return] beendet das Verschieben des Fensterkreuzes und nimmt die aktuelle Position als neue Position des Kreuzes an.

[Esc] beendet das Verschieben des Fensterkreuzes und läßt die alte Position des Kreuzes unverändert.

Wird eine bereits bestehende Trennlinie doppelt angeklickt, verschwindet die Linie komplett (schnelles Beseitigen von Unterfenstern). Entsprechend kann das Fensterkreuz dadurch beseitigt werden, daß seine Mitte doppelt angeklickt wird.

Die Einstellung der Unterfenster kann auch komplett über Tastatur erfolgen. Das ist dann etwa so aufwendig wie das Positionieren von Fenstern über Tastatur. Hierzu befindet sich im »Fenster«-Menü ein Befehl »Teilen...«. Wird dieser aufgerufen, erscheint ein graues Fensterkreuz am Bildschirm, das mit den Richtungstasten und weiteren Sondertasten auf eine neue Position gebracht werden kann. Das Beenden der Positionierung erfolgt mit der Eingabetaste. *Tabelle 1* beschreibt die einzelnen Möglichkeiten. Die Bewegung des Fensterkreuzes mit den Richtungstasten hat den Vorteil, daß sie in vertikalen und horizontalen Einheiten des Dokuments erfolgt (also etwa Linien und Spalten) und damit gegenüber dem linearen Verschieben mit der Maus in diesem Punkt überlegen ist. Das Fensterkreuz des Befehls »Teilen...« kann auch mit der Maus verschoben werden. Hierzu müssen Sie einfach nur die Maus in die gewünschte Richtung verschieben. Der Abschluß erfolgt durch Drücken der linken Maustaste.

Bestand bereits vorher ein Fensterkreuz, wird das neue Kreuz des Befehls »Teilen...« auf dieses bisherige Kreuz gesetzt. Damit kann man jenes leicht etwa um einige Zeilen oder Spalten verkleiner oder vergrößern (*Bild 6*). Bestand dagegen vorher noch kein Kreuz, wird mit vier gleich großen Unterfenstern gestartet. Mit wenigen Tastendrücken lassen sich daraus nur zwei Un-



terfenster machen oder sich die Unterfenster verkleinern. Das Tastatur-Interface wurde in seinen Einzelheiten von MS-Excel übernommen. Die CUA-Empfehlung beschreibt auch die Möglichkeit, mehr als vier Unterfenster zu implementieren. Sicherlich ist auch dies sehr reizvoll. Die Erweiterung wurde aber nicht im Rahmen dieses Artikels realisiert, da erst Erfahrungen mit der einfacheren Variante (bis zu vier Unterfenstern) gesammelt werden sollten. Dagegen wurden alle weiteren Interface-Vorschläge realisiert und es kam eine Menge Code zusammen ...

Mit [F6] kann der Fokus im Uhrzeigersinn zwischen Unterfenstern verschoben werden, mit zusätzlich [Shift] im Gegenuhrzeigersinn. Wie üblich, kann der Fokus auch mit der Maus verschoben werden.

Wenn Sie die Anzahl der Unterfenster zählen und die dazugehörigen Rolleisten, werden Sie verwundert feststellen, daß zu wenig Rolleisten vorhanden sind, um alle Unterfenster unabhängig voneinander zu bewegen. Es ist vielmehr so, daß eine Rolleiste sich auf alle Unterfenster bezieht, die entweder daneben oder darüber abgebildet sind. Somit konnte auf den Einbau von Rolleisten inmitten eines Fensters (ergonomisch sehr ungünstig) verzichtet werden. In *Bild 1* ist der Zusammenhang der Unterfenster durch die gestrichelten Linien kenntlich gemacht. Unterfenster lassen sich also nicht völlig frei voneinander im Dokument verschieben. In der Praxis hat die gleichzeitige Beeinflussung mehrerer Unterfenster durch eine Rolleiste aber mehr Vor- als Nachteile. In vielen Fällen möchte man beispielsweise zwei nebenanderliegende Unterfenster gleichzeitig von Zeile zu Zeile rollen. Als weiterer Nebeneffekt bleibt, daß sich die Kopfzeile und die Kopfspalte des Dokuments wie die gegenüberliegenden Rolleisten auf alle dazwischenliegenden Unterfenster beziehen. Es bleibt also eine eindeutige Zuordnung von Zeilen und Spalten gewahrt und es geht nicht übermäßig viel Platz innerhalb des Fensters für Verwaltungsobjekte (wie Rolleisten und Kopfinformation) verloren. Dies ist einer der Hauptvorteile von Unterfenstern gegenüber MDI-Fenstern. Bei letzteren ist alles mehrfach vorhanden, einschließlich des Fensterrahmens.

InitPaneManager initialisiert den Pane-Manager und erzeugt erforderliche Fensterklassen.
CreatePaneWindows erzeugt die erforderlichen Unterfenster für das Fenster, das vom Pane-Manager unterstützt werden soll und initialisiert weitere Daten.
ClosePanes beendet den Pane-Manager für das aktuelle Hauptfenster. Alle erzeugten Unterfenster werden wieder zerstört.
SetScrollValues initialisiert die Rolleisten für die Verwaltung der Unterfenster abhängig von Größenangaben in der Pane-Spezifikation.
SetPaneWindow setzt die Größen der Unterfenster nachdem diese verschoben wurden oder sich die Größe des Hauptfensters geändert hat.
ScrollPane erlaubt die Rollen oder Verschieben eine der Rolleisten und führt automatisch die erforderlichen Aktionen mit dem angezeigten Fensterinhalt in den angesprochenen Unterfenstern durch.
ChangePaneSize dient zur Ausführung des Befehls »Teilen...« und startet das Verändern der Unterfenster-Größen über Tastatur und Maus.
ProcessPaneMsg verarbeitet alle eingehenden Nachrichten an das Hauptfenster sofern sie den Pane-Manager betreffen und wird aus der Fensterfunktion des Hauptfensters aufgerufen.

DrawTrackPos zeichnet die aktuelle Position während des Ziehens des Rollfelds mit der Maus.
CreateDrawingTools erzeugt GDI-Attribute und -Objekte für das Zeichnen der verschiedenen Unterfenster mit *DrawPane*.
DestroyDrawingTools zerstört die mit *CreateDrawingTools* erzeugten GDI-Objekte nach dem Zeichnen mit *DrawPane*.
SetPaneFocus setzt oder entfernt die Anzeige im entsprechenden Unterfenster, wenn das Hauptfenster den Eingabefokus besitzt.
DrawPane zeichnet den geänderten Inhalt eines Unterfensters in den angegebenen Grenzen.

◄◄ *Tabelle 2: Eintrittsfunktionen in den Pane-Manager.*

◄ *Tabelle 3: Unterfenster-spezifische Funktionen der Applikation (werden vom Pane-Manager aufgerufen).*

Ein applikations-unabhängiger Unterfenster-Manager

Wie wir gesehen haben, sind Unterfenster vielfältig verwendbar, vielleicht gehören sie sogar wie die MDI-Schnittstelle in jede Applikation, mit der sich Dokumente bearbeiten lassen. Es ist daher sinnvoll, die Organisation der Unterfenster und die Kommunikation mit dem Benutzer unabhängig von der Applikation vorzunehmen und wenn möglich den Aufbau der Applikation nur an wenigen Stellen zu verändern, ähnlich wie dies auch bei meinem Hilfe-Manager [3], versucht wurde. Dies betrifft hier vorwiegend das Zeichnen der Unterfenster-Inhalte. Die Verwendung von Unterfenstern ist untrennbar mit Rolleisten beziehungsweise der Verschiebung der Anzeigeausschnitte über das Dokument verbunden. Da auch diese Implementierung ziemlich komplex und fehlerträchtig ist, ist es naheliegend, die Verwaltung von Blättern, Positionieren und Rollen in einen Pane-Manager zu verlagern. Im eigentlichen Applikations-Code müssen sich dann nur noch einige Funktionen befinden, die vom Pane-Manager mit den entsprechenden Parametern aufgerufen werden und beispielsweise den Fensterinhalt zeichnen oder den Eingabefokus korrekt anzeigen oder entfernen müssen. Ich muß gestehen, daß ich den Pane-Manager parallel zu der im folgenden vorgestellten Applikation entworfen habe. Es kann also durchaus sein, daß er noch Schwachstellen besitzt, die sich bei einer Portierung auf andere Applikationen bemerkbar machen. In diesem Fall müssen die Schnittstellen geändert oder erweitert werden.

Wie sehen diese Schnittstellen nun aus? Wie bereits beim Hilfe-Manager [3] wird ein Großteil der eingehenden Nachrichten an das Fenster mit den Unterfenstern managerspezifisch verarbeitet. Sie werden daher von der Fensterfunktion der Applikation alle an die Funktion `ProcessPaneMsg` übergeben und dort applikationsunabhängig analysiert. Einige Nachrichten werden vollständig »konsumiert«, so daß sie nicht mehr von der Applikations-Fensterfunktion weiterverarbeitet werden sollten. Andere müssen auch von dieser oder der Windows-Ersatzfunktion `DefWindowProc` verarbeitet werden. Die Funktion `Process-PaneMsg` umfaßt das komplette Benutzer-Interface zu den Rolleisten und die Veränderung des Fensterkreuzes und ähnliches. Lediglich das Tastatur-Interface für die Gestaltung des Fensterinhalts (Richtungstasten etc.) befindet sich (noch?) in der Fensterfunktion der Applikation.

Bevor aber der Pane-Manager seine Aufgaben reibungslos übernehmen kann, muß er erst einmal initialisiert werden. Dies erfolgt durch Aufruf der Funktion `InitPaneManager`. Im wesentlichen werden hier die Pane-spezifischen Fensterklassen erzeugt. Hierbei muß auf eine wichtige Tatsache hingewiesen werden: Unterfenster sind keine neuen (Tochter-) Fenster im Sinne von Windows. Dies hat sich beim Entwurf des Pane-Managers als wenig praktikabel herausgestellt. Es wird also beim Zeichnen eines Unterfensters nicht in ein neues Fenster gezeichnet, sondern weiterhin in das Fenster, das die »Unterfenster« besitzt. Lediglich die beiden Sprossen des Fensterkreuzes und die Bildschirmteiler links und über den Rolleisten sind neue Tochterfenster des Hauptfensters. Die gleiche Technik wurde auch bei MS-Excel angewandt.

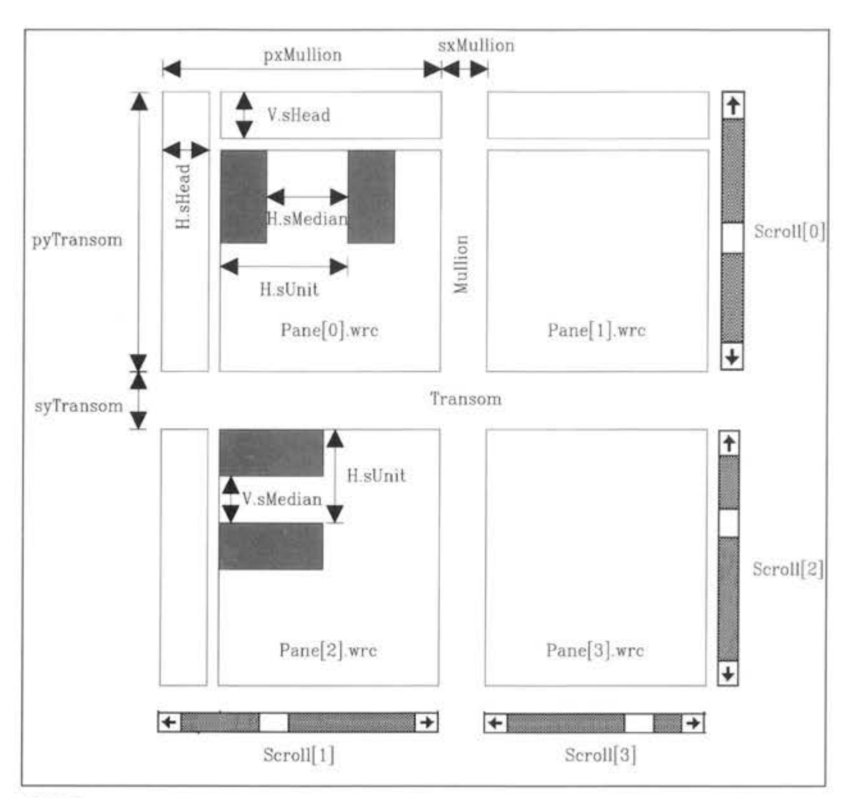


Bild 7: Elemente der Datenstruktur PANESPEC.

Der Pane-Manager kann mehrere Fenster gleichzeitig unterstützen. Dies ist wichtig, wenn er zusammen mit der MDI-Schnittstelle verwendet werden soll. Für jedes Fenster besitzt er eine Datenstruktur mit Namen `PANESPEC`, die alle fensterspezifischen Parameter enthält. Die Datenstruktur ist in `DUMP.H` definiert (siehe *Listing 4*). Anstatt die einzelnen Elemente der Struktur hier ausführlich zu beschreiben, sei auf *Bild 7* verwiesen, welche eine grafische Zuordnung der Struktur-Elemente zeigt. »Transom« und »Mullion« sind die englischen Fachbegriffe für waagrechte und senkrechte Fenstersprosse. Ein Dokument, in dem geblättert und gerollt werden kann, muß prinzipiell eine feste Einheitsgröße in vertikaler und horizontaler Richtung besitzen. Die Werte in `PANESPEC` beschreiben großteils solche Werte. Die `LONG`-Werte enthalten dokumentspezifische Anfangs- und Endwerte sowie die aktuelle Position als Index auf diese Einheiten. Alle Werte existieren für die Vertikale und die Horizontale. In einem Textverarbeitungs-Dokument könnte man sich als vertikale Einheit eine bestimmte Zeilenbreite und als horizontale Einheit eine standardisierte Zeichenbreite vorstellen. Bei einem Tabellenkalkulationsprogramm handelt es sich dagegen um die Höhe und Breite der einzelnen Felder. Wichtig ist, daß die Werte für das gesamte Dokument konstant sind. Dokumente können oben und links eine waagrechte Kopfzeile und eine senkrechte Kopfspalte besitzen, die jedoch nicht in die Unterfenster vervielfältigt werden. Die Dimensionen der beiden Köpfe sind ebenfalls in `PANESPEC angegeben`.

Bevor nun eine der weiteren Funktionen des Pane-Managers aufgerufen werden kann, muß ihm zunächst das aktuelle Fenster zugeordnet werden. Hierzu befindet sich in `PANESPEC` der Bezug des Fensters. Die Zuordnung erfolgt über die Zeigervariable `rCPS` (»current pane specification«), die jeweils von den einzelnen Funktionen des Managers verwendet wird. Danach werden durch Aufruf der Manager-Funktion `CreatePaneWindows` die Manager-Datenstrukturen und die benötigten Unterfenster für das ausgewählte Fenster angelegt. Mit `SetPaneWindow` wird anschließend die Größe der Unterfenster-Strukturen berechnet und der Fensterinhalt wird automatisch gezeichnet. Diese Funktion wird auch beim Verändern der Fenstergröße automatisch aufgerufen. Alles weitere übernimmt im folgenden der Pane-Manager, auch das Verschieben der Fensterinhalte oder des Fensterkreuzes. Lediglich bei Tasteneingaben, die ein Verschieben der Fensterinhalte zur Folge haben, muß die Funktion `ScrollPane` aufgerufen werden, die dann die erforderlichen Aktionen vom Korrigieren der Rollfeld-Position bis zum Neugestalten des Fensterinhalts vornimmt. Diese Funktion ermittelt auch, welche Teile des bisher angezeigten Fensters weiter benötigt werden und nur verschoben, aber nicht neugezeichnet werden müssen. Dadurch wird die Ausgabegeschwindigkeit etwa beim Rollen um eine Zeile oder Spalte deutlich erhöht. Soll das Pane-Management für ein Fenster beendet werden, wird die Funktion `ClosePanes` aufgerufen. Sie bewirkt, daß die Tochterfenster des Pane-Managers für das entsprechende Fenster wieder zerstört werden.

Zum Zeichnen bestimmter Datenstrukturen muß der Pane-Manager seinerseits auf applikationsspezifische Funktionen zugreifen. Hierzu übergibt er passende Parameter, die etwa das Neuzeichnen oder die Anzeige des Eingabefokus steuern. Die wichtigste Funktion für das Neuzeichnen des Fensterinhalts ist `DrawPane`. Sie wird für jedes zu zeichnende Unterfenster aufgerufen. Ihr werden bereits ein Zeichenbereich (*display context*), die Unterfensternummer und -größe übergeben. Der Pane-Manager übergibt dabei nicht die Koordinaten des Hauptfensters, in das eigentlich gezeichnet wird, sondern die Koordinaten des betreffenden Unterfensters. Die Position (0,0) ist also nicht zwingend die linke obere Ecke des Hauptfensters sondern die linke obere Ecke des angegebenen Unterfensters. Es ist daher für die Applikation verhältmäßig einfach, das Unterfenster unabhängig von dessen Position innerhalb des Hauptfensters immer gleichartig zu zeichnen. Es muß lediglich berücksichtigt werden, daß bestimmte Unterfenster auch noch den horizontalen oder vertikalen Kopf mitzeichnen müssen.

Im allgemeinen müssen GDI-Zeichenobjekte wie Strichbreiten, Farbenmuster oder Zeichensätze erzeugt werden. Damit dies nicht bei jedem Aufruf von `DrawPane` erneut erfolgen muß, wird dies einmalig vor einer Serie von `DrawPane`-Aufrufen durchgeführt, indem die Funktion `CreateDrawingTools` aufgerufen wird. Innerhalb von

`DrawPane` können die Objekte beliebig verändert oder zerstört werden, da sie vom Pane-Manager für jeden Aufruf von `DrawPane` gerettet werden. Nach der Rückkehr von `DrawPane` für das letzte Unterfenster werden dann die Objekte durch Aufruf der Applikationsfunktion `DestroyDrawingTools` wieder zerstört.

Die weiteren applikationsspezifischen Funktionen, die vom Pane-Mangager verwendet werden, erfüllen bestimmte Sonderaufgaben. Die Funktion `SetPaneFocus` setzt oder entfernt eine Kennung für den Eingabefokus im angegebenen Unterfenster. Dies kann beispielsweise die Einfügemarke (*Caret*) oder etwas ähnliches sein. Beim Ziehen eines Rollfelds innerhalb der Rolleiste muß oft die aktuelle Position angezeigt werden können. Hierfür dient die Funktion `DrawTrackPos`, die aufgerufen wird, sobald das Rollfeld mit gedrückter Maustaste verschoben wird. Damit ist der Pane-Manager bereits in groben Zügen besprochen. In den nächsten Kapiteln wird darauf eingegangen, wie er in der Praxis verwendet wird.

Eine Hex-Dump-Applikation als Anwendungsbeispiel

Als Beispiel, wie der Pane-Manager eingesetzt werden kann, wurde eine Applikation zum Anzeigen von beliebigen DOS-Dateien realisiert. Solche Applikationen existieren auch unter DOS oder sind in Debuggern wie CodeView als Befehle integriert. Sie zeigen die einzelnen Bytes einerseits als Hexwerte zwischen 00 und FF und zusätzlich als ASCII-Zeichen, wobei jeweils 16 Bytes zu einer Bildschirm-Zeile zusammengefaßt sind und davor die Adresse des ersten Bytes der jeweiligen Zeile steht. Bisher hat mich bei solchen Programmen immer gestört, daß man nie zwei Stellen einer Datei gleichzeitig ansehen kann, sondern entweder die Datei ausdrucken oder ständig hin- und herblättern muß. Außerdem fand ich die Darstellung teilweise etwas unübersichtlich, so daß man manchmal einzelne Bytes in der Zeile abzählen muß, um etwa eine bestimmte Adresse zu ermitteln. Außerdem störte mich insbesondere bei Programmen wie PCTools, daß man nur ziemlich umständlich an eine bestimmte Adresse gelangen kann. Grund genug, eine solche Applikation unter Windows zu entwickeln und für die gleichzeitige Anzeige zweier Dateiausschnitte den Pane-Manager zu verwenden. Damit dieser Artikel nicht zu lang wird, wurden dabei auf in der Praxis nützliche Teile wie Suchen oder Vergleichen von Daten und das Verändern von einzelnen Datenbytes erst einmal zurückgestellt. Die zur Erläuterung der Bedienung von Unterfenstern verwendeten *Bilder 3 bis 6* geben bereits einen Eindruck vom Erscheinungsbild der Applikation. Es wurden Linien zwischen einzelnen Abschnitten der Hex-

◄ *Listing 1: Das Hauptmodul der Applikation, DUMP.C.*

```
/*********************************************************************

 —— D U M P ——————— MS-Windows Application ——————————
                    Module DUMP.C

=====================================================================

This is the kernel module of the DUMP application.

_____________________________________________________________________

Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

Release 1.00 of 89-Sep-13 —— All rights reserved.

*********************************************************************/

/* read common header files */
#include "DUMP.H"

/*********************************************************************

 ———————————————— Application Variables ————————————————

*********************************************************************/

BYTE zAppName[]="DUMP"; /* application module name */
BYTE *rzAppTitle; /* pointer to application title name */
BYTE *rzNoMemory; /* pointer to error string "no memory" */
HANDLE hiMain;    /* handle to application instance */
HWND hwMain;     /* handle to main window */

FILESPEC FileSpec; /* file specification of dump-file */
int sxChar,syChar; /* size of system characters (fixed font size!) */
int nAddrChars=0;  /* number of used address field characters */
BYTE mcHex[]="0123456789ABCDEF";

PANESPEC PaneSpec;    /* application's pane specification */
PANESPEC *rCPS=NULL; /* pointer to current pane specification */

/* pointer to resources */
HCURSOR hcrWait;   /* hourglass */


/*********************************************************************

 ———————————————— Dialog-Box functions ————————————————

*********************************************************************/

/*********************************************************************
 f d A b o u t
=====================================================================

### Dialog Box function ###

This function processes any messages received by the "About"
dialog box.

Parameters:
 standard message data (see fwMain)

Return:
 standard dialog box function value
 DialogBox() returns TRUE.

*********************************************************************/

BOOL FAR PASCAL fdAbout(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {if (iMsg==WM_COMMAND)
   {/* OK pressed */
    EndDialog(hw,TRUE); return TRUE;
   }
  else if (iMsg==WM_INITDIALOG)
   {/* no focus is set */
    return TRUE;
   }
  else return FALSE;
 } /* fdAbout() */


/*********************************************************************
 S e t P a n e s
=====================================================================

This function sets the paning for the main window.
It determines and sets values which are used by the pane manager
during resizing, scrolling and drawing. The values are stored into the
structure <PaneSpec.H> for horizontal values and <PaneSpec.V> for
vertical values. If this function is explicitely called by the
application, the function SetPaneWindow() must be called to resize and
redraw the main window contents.

Parameters:
 none

Used Variables:
 PaneSpec

Return:
 none

*********************************************************************/

VOID SetPanes(VOID)

 {LONG s;
  RECT wrc;
  HDC hdc;
  TEXTMETRIC wtm;

  rCPS=&PaneSpec; /* set pointer to pane specification */
  /* ——— create system-dependent constants ——— */
  hdc=CreateIC("Display",NULL,NULL,NULL);
  SelectObject(hdc,GetStockObject(OEM_FIXED_FONT));
  GetTextMetrics(hdc,&wtm);
  sxChar=wtm.tmAveCharWidth; syChar=wtm.tmHeight;

  /* set fixed values of drawing specifications */
  PaneSpec.V.sUnit=wtm.tmHeight+2;
  PaneSpec.V.sMedian=wtm.tmDescent;
  PaneSpec.V.sHead=wtm.tmHeight+2;
  DeleteDC(hdc);
  PaneSpec.V.iMin=0;
  PaneSpec.V.iMax=max(FileSpec.vSize-1,0)/16; /* per line 16 bytes */
  PaneSpec.V.iPos=0; PaneSpec.V.iBase[0]=0;
```

► *Listing 1: (Fortsetzung)*

```
  /* set horizontal values */
  PaneSpec.H.iMin=0; PaneSpec.H.iMax=22;
  PaneSpec.H.iPos=0; PaneSpec.H.iBase[0]=0;
  PaneSpec.H.sUnit=(5*sxChar)/2;
  PaneSpec.H.sMedian=0;
  /* determine width of header from file size value */
  nAddrChars=4; /* at least 4 digits for address */
  s=FileSpec.vSize;
  while (s>=65536L)
   {/* increase size by one digit */
    nAddrChars++; s/=16;
   } /* while */
  PaneSpec.H.sHead=(nAddrChars+1)*sxChar; /* convert into units */
  CreatePaneWindows(); /* create pane windows for <rCPS> */
  /* set new application window sizes, redraw window contents */
  GetClientRect(hwMain,&wrc); SetPaneWindow(wrc.right,wrc.bottom);
 } /* SetPanes() */


/*********************************************************************
 C m d M a i n
=====================================================================

The command code <iCmd> of the main window is executed. Here only
command codes from the system menu (CMD_...) are executed.
Commands from child windows or similiar sources are not analysed here.

Parameters:
 iCmd  is the command code.

Return:
 TRUE if the command was consumed, FALSE if not.

*********************************************************************/

BOOL CmdMain(int iCmd)

 {HCURSOR hcrSave;
  BYTE z1[80],z2[80];

  switch(iCmd)
   {case CMD_ABOUT:
      /* ——— display about dialog box ——— */
      ReadDialog(DB_ABOUT,(FARPROC)fdAbout); return TRUE;
    case CMD_OPEN:
      /* ——— read name of file  ——— */
      if (!ReadDialog(DB_OPEN_FILE,fdOpenFile))
        return TRUE; /* cancelled */
      hcrSave=SetCursor(hcrWait); /* set hour glass */
      if (rCPS!=NULL)
       {/* ——— close old file, free memory ——— */
        CloseFile(&FileSpec);
        EnableMenuItem
         (GetMenu(hwMain),CMD_SPLIT,MF_BYCOMMAND|MF_GRAYED);
        ClosePanes(); /* pane manager not further valid */
       } /* if */
      /* ——— load specified file ——— */
      strcpy(zCurrentFile,zNewFile); /* set current file */
      FileSpec.hf=hfNewFile; /* set new handle */
      if (LoadFile(&FileSpec))
       {/* file header is loaded: change window text, set file spec */
        LoadString(hiMain,STFILETITLE,z1,sizeof(z1));
        sprintf(z2,z1,zCurrentFile); SetWindowText(hwMain,z2);
        SetPanes(); /* set new size of window-contents, redraw it */
        EnableMenuItem
         (GetMenu(hwMain),CMD_SPLIT,MF_BYCOMMAND|MF_ENABLED);
       }
      else
       {/* cannot load file: close it again */
        CloseFile(&FileSpec);
        SetWindowText(hwMain,rzAppTitle); /* set standard text */
        InvalidateRect(hwMain,NULL,TRUE); /* clear window) */
       } /* if */
      SetCursor(hcrSave); /* set old cursor */
      return TRUE;
    case CMD_SPLIT:
      /* change size of panes */
      ChangePaneSize();
   } /* switch */
  return FALSE; /* not consumed */
 } /* CmdMain() */


/*********************************************************************

 ———————————————— Drawing Functions ————————————————

*********************************************************************/

/* application specific drawing tools */
HPEN hpnDot;
HPEN hpnLine;
HBRUSH hbrBkgr;


/*********************************************************************
 D r a w T r a c k P o s
=====================================================================

This function draws the preliminary value of the address which is
determined by moving the holding-down scroll bar thumb. The value is
written into the first line of the horizontal header in pane 0 or
pane 2. If the value is different from the current set position, it is
displayed inverted. If it is the same value it is not changed. The
function draws the value immediately.

Parameters:
 iTrackBase  is the current tracking base value.

Used Globals:
 rCPS

Return:
 none

*********************************************************************/

VOID DrawTrackPos(int iPane,LONG iTrackBase)

 {HDC hdc;
  BOOL bOrg; /* TRUE if original location */
  RECT wrcBkgr;
  BYTE z[10+1];
  int i;

  /* determine DC, determine drawing point */
  hdc=GetDC(hwMain); SelectObject(hdc,GetStockObject(OEM_FIXED_FONT));
  bOrg=iTrackBase==rCPS->V.iBase[iPane>1];
  SelectObject(hdc,GetStockObject(OEM_FIXED_FONT));
  SetTextColor(hdc,GetSysColor(bOrg? COLOR_WINDOWTEXT:COLOR_WINDOW));
  SetBkColor(hdc,GetSysColor(bOrg? COLOR_WINDOW:COLOR_WINDOWTEXT));
```

Tabelle eingefügt, um die Ausgabe übersichtlicher zu gestalten. Ferner wurden die Abstände zwischen den Byte-Werten auf halbe Leerzeichen reduziert. Die Ausgabe der Daten als Zeichen erfolgen nicht im ANSI-Zeichensatz, sondern im IBM-Zeichensatz, da die meisten Dateien, die man sich ansehen möchte, wohl diesen Zeichensatz verwenden. Bei einer »professionellen« Version der Applikation müßte man natürlich mehr Möglichkeiten anbieten. Auch wäre hierbei die MDI-Schnittstelle [2] zu integrieren. Der Pane-Manager ist ja bereits vorbereitet, Unterfenster von mehreren Fenstern gleichzeitig zu verwalten.

Listing 1 zeigt den Kern der Applikation. Dieses Modul DUMP.C enthält die Fensterfunktion der Applikation, die Interpretation der Menübefehle und die Initialisierung der Applikation. All dies ist weitgehend Windows-Standard. Weiterhin existieren mehrere Funktionen zum Zeichnen des Fensterinhalts, die aber von dem Pane-Manager aufgerufen werden. Dieser befindet sich applikationsunabhängig in dem Modul PANE.C, das in *Listing 2* abgebildet ist. Dieses Modul wird im nächsten Abschnitt erläutert. Die dritte Quelldatei heißt FILE.C, ist in *Listing 3* ausgedruckt und enthält die Anzeige des Dialogfelds zur Eingabe des Dateinamens und die gesamte Verwaltung des Dateizugriffs einschließlich einer virtuellen Pufferverwaltung. Gemeinsame Deklarationen und Funktionsprototypen aller Quelldateien stehen in DUMP.H in *Listing 4*. Hier befindet sich auch die Schnittstelle des Pane-Managers, etwa die Beschreibung der Struktur `PANESPEC`. Die Datei DEFS.H im *Listing 9* enthält wie üblich gemeinsame Definitionen von Quellcode und Ressourcen-Datei, deren Inhalt in *Listing 7* ausgedruckt ist. Die Definition der beiden Dialogfelder für die Applikations-Beschreibung und die Eingabe des Dateinamens befinden sich in DUMP.DLG, ausgedruckt in *Listing 8*. Die restlichen Listings erzeugen zwei verschiedene Version des Programms, dazu weiter unten.

Details zur Realisierung des Unterfenster-Managers

Innerhalb des Pane-Managers wurden einige interessante Windows-Funktionen aufgerufen, deren Anwendung einige Bemerkungen erfordern. Es kann aber nicht auf jedes Detail des Managers eingegangen werden. Hierfür müssen die Listings eingehend studiert werden, insbesondere das Modul `PANE.C`.

Als erstes soll die Verarbeitung von `WM_PAINT` in der Funktion `ProcessPaneMsg` besprochen werden. Sie muß die Zuordnung zum Neuzeichnen der einzelnen Unterfenster mit Hilfe der Applikationsfunktion `DrawPane` vornehmen. Doch zunächst werden die applikationsspezifischen GDI-Objekte durch Aufruf der Funktion CreateDrawingTools erzeugt. Diese werden jetzt für

jeden DrawPane-Aufruf mit Hilfe der Windows-Funktion SaveDC gerettet und nach dem Aufruf mit RestoreDC wieder unverändert zurückgeholt. Somit können diese Objekte beliebig innerhalb von DrawPane verändert werden, eine durch CreateDrawingTools eingestellte Initialisierung bleibt unverändert. Als nächstes muß der Zeichenausschnitt, dessen Koordinaten sich natürlich auf das Hauptfenster und nicht die Unterfenster beziehen, auf das betreffende Unterfenster reduziert werden. Dies geschieht durch Berechnung der Schnittmenge des Zeichenausschnitts und des jeweiligen Unterfensters mittels der Windows-Funktion IntersectClipRect. Nur wenn tatsächlich dann noch ein Zeichenausschnitt besteht, wird DrawPane aufgerufen. Vorher wird aber noch mit der Windows-Funktion SetViewportOrg das Koordinatensystem so verschoben, daß der Ursprung zum Zeichnen mit dem Ursprung des betreffenden Unterfensters übereinstimmt. Sind alle Unterfenster gezeichnet, werden die erzeugten GDI-Objekte durch Aufruf von DestroyDrawingTools wieder zerstört.

Der weitere Code in ProcessPaneMsg dient in erster Linie dazu, den Pane-Manager betreffende Tasteneingaben korrekt zu verarbeiten. So wechselt die Taste VK_F6 etwa den Fokus zwischen den einzelnen Unterfenstern im Uhrzeiger- oder Gegenuhrzeigersinn. Tastendrücke während des Verschiebens des Fensterkreuzes werden dagegen in der Fensterfunktion fwSplit der Bildschirmteiler verarbeitet.

Diese Funktion ist eines der »seitenlangen« Unterprogramme des Pane-Managers. In ihr werden alle Vorgänge verarbeitet, die das Verändern des Fensterkreuzes betreffen. Die Bildschirmteiler-Fenster können dabei verschiedene Zustände einnehmen, die durch static-Variablen innerhalb der Fensterfunktion festgelegt sind. In der Fensterfunktion folgt auch die Verarbeitung des Umpositionierens und des schließlichen Neuzeichnens des Fensterkreuzes. Während des Positionierens (mit Maus oder Tastatur) wird das vorläufige Fensterkreuz als graue Balken dargestellt. Diese Balken invertieren den Untergrund, der in diesem Fall dem Hauptfenster, also dem Mutterfenster (*parent window*) des Bildschirmteilers gehört. Hierzu wird die GDI-Funktion PatBlt verwendet. Die Invertierung des Untergrunds statt dessen Überzeichnen hat den Vorteil, daß das Hauptfenster beim Verschieben des Kreuzes nicht neu gezeichnet werden muß und aus dem Grund das Verschieben sehr schnell vonstatten geht.

Eine interessante Funktion unabhängig vom Pane-Manager stellt ScrollPane dar. Sie muß ähnlich auch bei anderen Applikationen realisiert werden, auch dann, wenn keine Unterfenster verwendet werden sollen. In unserem Fall ist sie natürlich durch die Anwesenheit mehrerer Unterfenster noch etwas komplizierter geworden. Eine Aufgabe dieser Funktion besteht zunächst in der

◀ *Listing 1: (Fortsetzung)*

```
  wrcBkgr.left=rCPS->Pane[0].wrc.left;
  wrcBkgr.right=wrcBkgr.left+rCPS->H.sHead-1;
  wrcBkgr.top=rCPS->Pane[iPane].wrc.top+(iPane>1? 0:rCPS->V.sHead);
  wrcBkgr.bottom=wrcBkgr.top+rCPS->V.sUnit-1;
   {/* -- draw hex number of address -- */
    LONG v=iTrackBase;
    z[nAddrChars-1]='0'; /* last digit */
    for (i=nAddrChars-2;i>=0;i--) {z[i]=mcHex[v&0xF]; v>>=4;}
    ExtTextOut
     (hdc,wrcBkgr.left+sxChar/2,wrcBkgr.top,
      ETO_OPAQUE,&wrcBkgr,z,nAddrChars,NULL
     );
   } /* block */
  ReleaseDC(hwMain,hdc);
  if (bOrg)
   {/* redrawing not needed: invalidate rectangle */
    rCPS->wrcTrackRedraw.left=rCPS->wrcTrackRedraw.right;
   }
  else
   {/* tracking field must be redrawn: set rectangle */
    rCPS->wrcTrackRedraw=wrcBkgr;
   } /* if */
 } /* DrawTrackPos() */


/*****************************************************************
 C r e a t e D r a w i n g T o o l s
=================================================================

This function creates GDI tools which are needed for the repainting of
the main window contents by the function DrawPane. The default values
are set.

Parameters:
 hdc  is the device context for the drawing.

Return:
 none

*****************************************************************/

VOID CreateDrawingTools(HDC hdc)

 {/* create needed drawing tools */
  hpnDot=CreatePen(PS_DOT,1,GetSysColor(COLOR_WINDOWTEXT));
  hpnLine=CreatePen(PS_SOLID,1,GetSysColor(COLOR_WINDOWTEXT));
  hbrBkgr=CreateSolidBrush(GetSysColor(COLOR_WINDOW));
  /* set drawing tools */
  SelectObject(hdc,GetStockObject(OEM_FIXED_FONT));
  SetTextColor(hdc,GetSysColor(COLOR_WINDOWTEXT));
  SelectObject(hdc,hpnLine); SelectObject(hdc,hbrBkgr);
 } /* CreateDrawingTools() */


/*****************************************************************
 D e s t r o y D r a w i n g T o o l s
=================================================================

This function destroyes all GDI tools which are created by the
preceded call of CreateDrawingTools().

Parameters:
 none

Return:
 none

*****************************************************************/

VOID DestroyDrawingTools(VOID)

 {DeleteObject(hpnDot); DeleteObject(hpnLine); DeleteObject(hbrBkgr);
 } /* DestroyDrawingTools() */


/*****************************************************************
 S e t P a n e F o c u s
=================================================================

The focus is switched to pane <iPane> or destroyed if <iPane> is -1.
This function should set to caret to the specified pane or destroy the
caret.

Parameters:
 iPane .. is the index of a valid pane which contains the focus. This
          can be a value of 0..3. If the value is -1, the application
          window loses the input focus.
 bChange  is TRUE if the focus is changed between the panes without
          changing the focus of the application and FALSE if the
          application has got or lost the focus.

Used Globals:
 Pane

Return:
 none

*****************************************************************/

VOID SetPaneFocus(int iPane,BOOL bChange)

 {POINT pCaret;

  if (iPane<0)
   {/* hide caret (focus lost), return */
    DestroyCaret(); return;
   } /* if */

  /* calculate caret position in pane (after 1st char. in 1st line) */

  pCaret.x=rCPS->Pane[iPane].wrc.left+sxChar/4;
  if (!(iPane&1)) pCaret.x+=rCPS->H.sHead; /* add vertical header */
  pCaret.y=rCPS->Pane[iPane].wrc.top+syChar;
  if (iPane<2) pCaret.y+=rCPS->V.sHead; /* add horizontal header */
  if (!bChange)
   {/* create caret */
    CreateCaret(rCPS->hwPane,NULL,sxChar,rCPS->V.sMedian);
   } /* if */
  /* set caret position */
  SetCaretPos(pCaret.x,pCaret.y);
  if (!bChange) ShowCaret(rCPS->hwPane); /* display caret */
 } /* SetPaneFocus() */


/*****************************************************************
 D r a w P a n e
=================================================================

This function redraws one of the available panes of the main window.
At most 4 panes can be exist in the following order:
```

▸ *Listing 1: (Fortsetzung)*

```
+-----+-----+-+
|     |     |^|   pane 0 exists always.
|  0  |  1  |0|   pane 1 exists only if rCPS->pxMullion is non-0.
|     |     |v|
+-----+-----+-+   pane 2 exists only if rCPS->pyTransom is non-0.
|     |     |^|
|  2  |  3  |2|
|     |     |v|   pane 3 exists only if rCPS->pxMullion and
+-----+-----+-+   rCPS->pyTransom are both non-0.
|<-1->|<-3->|
+-----+-----+

Parameters:
 hdc ... is the display context for drawing the client area of the
         main window.
 iPane   is the index of the pane in range 0..3.
 sxPane  is the width of the pane.
 syPane  is the height of the pane.

Used Globals:
 rCPS

Return:
 none

*********************************************************************/

VOID DrawPane(HDC hdc,int iPane,WORD sxPane,WORD syPane)
 {int vV,vH;
  int iV,iH,i,s,sl,iWrt;
  int iVStart,nLines;
  int iHStart,iHEnd,iHBase;
  HANDLE hmgData; /* handle of global memory block */
  BYTE FAR *rd; /* pointer to global memory data */
  RECT wrcClip,wrcLines;
  POINT pBase,pStart,pDraw;
  int pyTerm=0; /* terminator line location */
  LONG vAddr,vA;
  BYTE z[60+1];
  int mv[60+1];

  GetClipBox(hdc,&wrcClip);
  /* set pane base values */
  pBase.x=iPane&1? 0:rCPS->H.sHead; pBase.y=iPane>1? 0:rCPS->V.sHead;
  /* nothing to draw => return */
  if (wrcClip.left>pBase.x+23*rCPS->H.sUnit) return;
  iHBase=(int)rCPS->H.iBase[iPane&1]; /* index for first column */
  /* ---- determine values for valid vertical drawing area ---- */
  iVStart=(max(wrcClip.top,pBase.y)-pBase.y)/rCPS->V.sUnit; /* line */
  vAddr=(rCPS->V.iBase[iPane>1]+iVStart); /* address of first line */
  pStart.y=pBase.y+iVStart*rCPS->V.sUnit; /* drawing start point */
  /* ---- determine number of lines to draw ---- */
  nLines=(wrcClip.bottom-pBase.y+rCPS->V.sUnit-1)/
         rCPS->V.sUnit-iVStart;
  if (nLines<=0) nLines=1; /* at least one line */
  if (vAddr+(nLines-1)>rCPS->V.iMax)
     nLines=(int)(rCPS->V.iMax-vAddr)+1;
  if (!(iPane&1) && wrcClip.left<rCPS->H.sHead)
   {/* ===== draw vertical header ===== */
    LONG vA=vAddr;
    if (iPane<2 && wrcClip.top<rCPS->V.sHead)
     {/* draw headline for vertical header */
      int s;
      s=sprintf
       (z,nAddrChars<5? "Addr":nAddrChars<8? "Addr.":"Address");
      ExtTextOut(hdc,((nAddrChars-s+1)*sxChar)/2,0,0,NULL,z,s,NULL);
     } /* if */
    pDraw.y=pStart.y; pDraw.x=sxChar/2;
    for (iV=0;iV<nLines;iV++)
     {/* -- draw hex number of address -- */
      LONG v=vA;
      z[nAddrChars-1]='0'; /* last digit */
      for (i=nAddrChars-2;i>=0;i--) {z[i]=mcHex[v&0xF]; v>>=4;}
      ExtTextOut(hdc,pDraw.x,pDraw.y,0,NULL,z,nAddrChars,NULL);
      pDraw.y+=rCPS->V.sUnit; vA++;
     } /* for */
    /* -- draw vertical header separator line -- */
    pDraw.x=rCPS->H.sHead-1;
    pDraw.y=min(pStart.y+nLines*rCPS->V.sUnit,wrcClip.bottom);
    MoveTo(hdc,pDraw.x,wrcClip.top); LineTo(hdc,pDraw.x,pDraw.y);
   } /* if */
  /* ---- exclude header from line clipping area ---- */
  wrcLines=wrcClip; /* set line drawing rectangle */
  if (wrcLines.left<pBase.x) wrcLines.left=pBase.x;
  /* ===== define space separation for ExtTextOut() ===== */
  for (i=32;i<40;i++) mv[i]=sxChar;
  memcpy(mv+40,mv+32,8*sizeof(int));
  mv[39]=3*(sxChar/2); /* separation between 8 characters */
  if (iPane<2 && wrcClip.top<rCPS->V.sHead)
   {/* ===== draw horizontal headline ===== */
    /* -- change ExtTextOut()-spacing -- */
    for (i=0;i<16;i++) mv[i]=rCPS->H.sUnit;
    mv[15]=rCPS->H.sUnit-(sxChar+2)/4; /* separation */
    memcpy(mv+16,mv+32,16*sizeof(int));
    /* -- create hex code byte for headline -- */
    for (i=0;i<16;i++) z[i]=mcHex[i];
    memcpy(z+16,mcHex,16); /* set 0..9A..F */
    /* draw complete line */
    pDraw.x=pBase.x-iHBase*rCPS->H.sUnit+3*(sxChar+2)/4+1;
    ExtTextOut(hdc,pDraw.x,0,ETO_CLIPPED,&wrcLines,z,32,mv);
   } /* if */
  /* ===== draw contents of lines: hex values characters ===== */
   {int *ri=mv;
    int vd=rCPS->H.sUnit-sxChar;
    /* -- change ExtTextOut()-spacing -- */
    for (i=0;i<16;i++) {*ri++=sxChar; *ri++=vd;}
    *(ri-1)=rCPS->H.sUnit-sxChar/2-(sxChar+2)/4; /* separation */
   } /* block */
  /* determine drawing location */
  pDraw.x=pBase.x-iHBase*rCPS->H.sUnit+(sxChar+2)/4+1;
  pDraw.y=pStart.y; vA=16*vAddr;
  hmgData=NULL; /* no buffer selected */
  for (iV=0;iV<nLines;iV++)
   {/* ---- draw one hex code line ---- */
    int iHex=0;
    int iChar=iHex+32;
    if (!hmgData && vA<FileSpec.vSize)
     {/* new data block must be read from file */
      hmgData=ReadDataBlock(&FileSpec,vA);
      if (!hmgData) break; /* memory error => exit loop */
      rd=GlobalLock(hmgData);
      rd+=(WORD)vA&0xFFF; /* to current read location */
     } /* if */
    for (i=0;i<16;i++)
     {/* draw single hex code byte */
      BYTE d;
      /* end of file reached => break */
      if (vA>=FileSpec.vSize)
       {/* end of file: set spaces instead characters */
        z[iHex++]=' '; z[iHex++]=' '; z[iChar++]=' ';
       }
```

Ermittlung der Auswirkungen der übergebenen Rolleisten-Operation. Beim Blättern wird übrigens gemäß CUA-Vorschrift die letzte Zeile als die erste im nächsten Blatt beziehungsweise umgekehrt dargestellt. Es wird also nur um (n-1) Zeilen geblättert, damit die Zuordnung zur vorangegangenen Seite klarer wird. Aus den festgestellten Auswirkungen wird dann die neue Dokument-Position ermittelt und mit der alten verglichen. Aus dem Ergebnis dieses Vergleichs wird dann abgeleitet, ob das betroffene Unterfenster komplett neu gezeichnet werden muß, oder ob durch Verschieben bereits gezeichneter Teile die Zeichenarbeit reduzieren kann. Der Algorithmus ist davon unabhängig, ob nur eine Zeile oder Spalte gerollt oder größere Bereiche verschoben werden müssen: Was am Bildschirm bereits steht, braucht nicht mehr neu gezeichnet zu werden. Für das Verschieben ist die mit Windows 2.0 neu eingeführte Funktion `ScrollDC` zuständig, die auch gleich das Rechteck des Bereichs zurückgibt, der neu gezeichnet werden muß. Dabei muß man aber aufpassen: Überlappen sich alte und neue Position eines Bereichs an keiner Stelle, muß auch der Bereich zwischen beiden Positionen neu gezeichnet werden. Dies berücksichtigt leider `ScrollDC` nicht, so daß man beim Verschieben über größere Bereiche merkwürdige Resultate entstehen. Deshalb findet nach Aufruf von `ScrollDC` eine Korrekturberechnung des von `ScrollDC` zurückgegebenen Rechtecks statt.

Die weiteren verwendeten Algorithmen im Pane-Manager sind zwar teilweise recht komplex, aber besonders aufregende Dinge im Zusammenhang mit Windows sind nicht mehr erwähnenswert. Aus dem Grund soll an dieser Stelle die Beschreibung des Pane-Managers auch beendet werden. In den folgenden Kapiteln werden noch auf interessante Details im Zusammenhang mit der Applikation DUMP eingegangen.

Ausgabe des Hex-Dumps

Die DUMP-Applikation erlaubt nicht die Veränderung des angezeigten Dokuments. Um dennoch zu wissen, wo der »Eingabe«-Fokus steht, mit dem ja über die Tastatur gerollt und geblättert werden kann, wird mit der Funktion `Set–PaneFocus` die Einfügemarke fokusabhängig in einem der Unterfenster angezeigt, allerdings immer an derselben Stelle.

Die Ausgabe des Fensterinhalts beschränkt sich im wesentlichen auf die Funktion `DrawPane`. Sie zeichnet jeweils den geänderten Inhalt für ein Unterfenster, wobei sie gegebenenfalls auch die entsprechenden vertikalen und horizontalen Köpfe mitzeichnet. Dies erfolgt natürlich nur bei den Unterfenstern, die diese Köpfe besitzen. Zu deren Bestimmung wird der Parameter *iPane* von `DrawPane` verwendet, der angibt, welches Unter-

fenster gerade gezeichnet werden soll. Die Zeichenfunktion umfaßt nur den Bereich, der nicht durch Verschieben von bereits angezeigten Teilen – etwa beim Rollen – angezeigt werden kann. Es brauchen für das Zeichnen daher nur die Daten aufbereitet zu werden, die sich auch tatsächlich in dem entsprechenden Zeichenausschnitt (im Original *clipping region*) befinden. Hiervon wird in der Funktion ausgiebig Gebrauch gemacht, um das Zeichnen möglichst schnell zu gestalten. Das Rechteck, das alle gesetzten Zeichenausschnitte umschließt, kann mit Hilfe der Windows-Funktion `GetClipBox` ermittelt werden. Die einzelnen Zeichenausschnitte können natürlich komplizierter als ein Rechteck sein – sie müssen noch nicht einmal zusammenhängen. Das mit `GetClipBox` ermittelte Rechteck kann also größer sein als der tatsächliche Zeichenbereich. Dies ist jedoch in der Praxis von geringer Bedeutung, was im nächsten Abschnitt klarer werden wird. Liegt aber etwa eine Ausgabezeile völlig außerhalb des Zeichenausschnitts, ist es sinnvoll dies zu erkennen und dann auch keine Ausgaben zu erzeugen. Beim DUMP-Programm betrifft dies beispielsweise den horizontalen Kopf wenn geblättert wird. Ähnlich verhält es sich mit vertikalen Linien, die natürlich nicht gezeichnet werden müssen, wenn sie ebenfalls nirgends in den Zeichenbereich hineinragen. Es ist oft sehr schnell ermittelt, ob eine Linie oder ein rechteckähnliches Objekt in den Zeichenausschnitt hineinragt. Zwar »bemüht« sich der Windows-GDI, Elemente außerhalb des Zeichenausschnitts »sehr schnell nicht zu zeichnen«, aber eine gewisse Verzögerung läßt sich dennoch nicht vermeiden. Dies betrifft insbesondere Objekte wie Ellipsen, Polygone und ähnliches, bei denen es einen erheblichen Aufwand darstellt, herauszufinden, welche Teilelemente zu berechnen sind, weil sie teilweise gezeichnet werden müssen und welche Teilelemente nicht berechnet werden müssen, weil sie überhaupt nicht dargestellt werden müssen.

Man kann sich bei der Verwendung der Zeichenausschnitt-Informationen auch »kaputt-optimieren«, ähnlich wie beim Übersetzerbau. Die beiden folgenden Ratschläge wurden daher auch aus diesem Zweig der Informatik auf die Optimierung von Zeichenoperationen übertragen:

• Nehmen Sie nur dann Optimierungen vor, wenn Sie auch sicher sind, daß sie die korrekte Ausgabe der Zeichnung nicht beinflussen. Eine schnelle Ausgabe eines Zeichenbereichs, in dem einige Objekte fehlen, weil der Optimierungsalgorithmus sie »übersehen« hat, ist für den Anwender sehr lästig, vor allem, wenn es wiederholt vorkommt. Machen Sie sich klar, wie Zeichenausschnitte aussehen können. Verwenden Sie nur das mit `GetClipBox` ermittelte Rechteck zur Bestimmung des Zeichenausschnitts oder die präziseren aber auch langsameren Funktionen `PtVisible` und `RectVisible`.

◄ *Listing 1: (Fortsetzung)*

```
      else
       {/* set hex- and character-value */
        d=*rd++; vA++;
        z[iHex++]=mcHex[d>>4]; z[iHex++]=mcHex[d&0xF];
        z[iChar++]=d; /* character code */
       } /* if */
     } /* for */
    if (((WORD)vA&0xFFF)==0)
     {/* new data block must be read */
      GlobalUnlock(hmgData); hmgData=NULL;
     } /* if */
    /* draw complete line */
    ExtTextOut(hdc,pDraw.x,pDraw.y,ETO_CLIPPED,&wrcLines,z,iChar,mv);
    pDraw.y+=rCPS->V.sUnit; /* next line */
   } /* for */
  if (hmgData) GlobalUnlock(hmgData);
  if (vA>=FileSpec.vSize)
   {/* terminating separator line must be drawn: set location */
    pyTerm=pDraw.y;
   } /* if */
  /* ===== draw vertical separator lines ===== */
  iHStart=(max(wrcClip.left,pBase.x)-pBase.x)/rCPS->H.sUnit+iHBase;
  iHEnd=(wrcClip.right-pBase.x+rCPS->H.sUnit-1)/rCPS->H.sUnit+iHBase;
  SelectObject(hdc,hpnDot);
  pDraw.y=min(pStart.y+nLines*rCPS->V.sUnit,wrcClip.bottom);
  if (iHStart<=8 && iHEnd>8)
   {pDraw.x=pBase.x+(8-iHBase)*rCPS->H.sUnit;
    MoveTo(hdc,pDraw.x,wrcClip.top); LineTo(hdc,pDraw.x,pDraw.y);
   } /* if */
  if (iHStart<=16 && iHEnd>16)
   {pDraw.x=pBase.x+(16-iHBase)*rCPS->H.sUnit;
    MoveTo(hdc,pDraw.x,wrcClip.top); LineTo(hdc,pDraw.x,pDraw.y);
   } /* if */
  if (iHStart<20 && iHEnd>19)
   {pDraw.x=pBase.x+(19-iHBase)*rCPS->H.sUnit+(5*sxChar+1)/4;
    MoveTo(hdc,pDraw.x,wrcClip.top); LineTo(hdc,pDraw.x,pDraw.y);
   } /* if */
  SelectObject(hdc,hpnLine);
  if (iHStart<=23 && iHEnd>23)
   {pDraw.x=pBase.x+(23-iHBase)*rCPS->H.sUnit;
    MoveTo(hdc,pDraw.x,wrcClip.top); LineTo(hdc,pDraw.x,pDraw.y);
   } /* if */
  pDraw.x=min(pBase.x+(23-iHBase)*rCPS->H.sUnit,wrcClip.right);
  if (iPane<2 && wrcClip.top<rCPS->V.sHead)
   {/* ===== draw horizontal header separator line ===== */
    MoveTo(hdc,wrcClip.left,rCPS->V.sUnit-2);
    LineTo(hdc,pDraw.x,rCPS->V.sUnit-2);
   } /* if */
  if (pyTerm)
   {/* ===== draw terminator line ===== */
    MoveTo(hdc,wrcClip.left,pyTerm);
    LineTo(hdc,pDraw.x+1,pyTerm);
   } /* if */
  /* ===== draw dotted horizontal separator lines ===== */
  iV=8-((int)vAddr&7); if (iV==0) iV=8;
  pStart.y+=iV*rCPS->V.sUnit-1; /* vertical point of first line */
  SelectObject(hdc,hpnDot);
  for (;iV<=nLines;iV+=8)
   {MoveTo(hdc,wrcClip.left,pStart.y); LineTo(hdc,pDraw.x,pStart.y);
    pStart.y+=8*rCPS->V.sUnit; /* next line position */
   } /* for */
 } /* DrawPane() */


/**********************************************************************

 ------------------------ Main Window function ------------------------

**********************************************************************/

LONG FAR PASCAL fwMain(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {LONG vRet;

  if (rCPS)
   {/* call pane-manager's message processor */
    vRet=ProcessPaneMsg(hw,iMsg,uP1,ulP2);
    if (vRet) return vRet; /* consumed */
   } /* if */
  switch (iMsg)
   {case WM_CREATE:
      /* set application window handle */
      hwMain=hw; PaneSpec.hwPane=hw;
      return 0L;

    case WM_KEYDOWN:
     {BOOL bCtrl=GetKeyState(VK_CONTROL)>>15;
      BOOL bVert=!bCtrl;
      int  uScrollCmd;
      static BYTE miVScroll[]={0,0,2,2};
      static BYTE miHScroll[]={1,3,1,3};
      switch (uP1)
       {case VK_PRIOR:
          /* page up/left */
          uScrollCmd=SB_PAGEUP; break;
        case VK_NEXT:
          /* page down/right */
          uScrollCmd=SB_PAGEDOWN; break;
        case VK_HOME:
          /* start of line/data */
          uScrollCmd=SB_TOP; bVert=!bVert;
          break;
        case VK_END:
          /* end of line/data */
          uScrollCmd=SB_BOTTOM; bVert=!bVert;
          break;
        case VK_UP:
          /* line up */
          uScrollCmd=SB_LINEUP; bVert=TRUE;
          break;
        case VK_DOWN:
          /* line down */
          uScrollCmd=SB_LINEDOWN; bVert=TRUE;
          break;
        case VK_LEFT:
          /* line left */
          uScrollCmd=SB_LINEUP; bVert=FALSE;
          break;
        case VK_RIGHT:
          /* line right */
          uScrollCmd=SB_LINEDOWN; bVert=FALSE;
          break;
        default:
          goto DEFAULT; /* not consumed */
       } /* switch */
      /* execute scrolling */
      ScrollPane
       (bVert? miVScroll[rCPS->iFocusPane]:
               miHScroll[rCPS->iFocusPane],uScrollCmd,0);
      return 0L; /* consumed */
     } /* case */
```

▸ Listing 1: (Fortsetzung)

```
    case WM_COMMAND:
      if (uP1!=CMD_EXIT)
       {/* user control command */
        CmdMain(uP1);
        break; /* not consumed */
       } /* if */
      /* continue */
    case WM_CLOSE:
      DestroyWindow(hw); return 0L;
    case WM_DESTROY:
      PostQuitMessage(0); return 0L;
   } /* switch */
  DEFAULT:
  return DefWindowProc(hw,iMsg,uP1,ulP2);
 } /* fwMain() */


/*********************************************************************

 —— Window Dependent Initialization (One for All Instances) ——

*********************************************************************/

/*********************************************************************
 C r e a t e C l a s s
=====================================================================

The classes of all application specified windows are registered.
Following classes exists:

Parameters:
 none

Return:
 TRUE   if all classes created.
 FALSE  if any error during creation (memory error).

*********************************************************************/

BOOL CreateClass(VOID)

 {WNDCLASS wwc;

  /* —— create window class of application —— */
  wwc.lpszClassName=zAppName; wwc.hInstance=hiMain;
  wwc.lpfnWndProc=fwMain;
  wwc.hCursor=LoadCursor(NULL,IDC_ARROW);
  wwc.hbrBackground=GetStockObject(WHITE_BRUSH);
  wwc.style=0; /* CS_HREDRAW or CS_VREDRAW not needed! */
  wwc.hIcon=LoadIcon(hiMain,MAKEINTRESOURCE(ICO_MAIN));
  wwc.lpszMenuName=MAKEINTRESOURCE(MNU_MAIN);
  wwc.cbClsExtra=0; wwc.cbWndExtra=0;
  /* register window, return if error */
  if (!RegisterClass(&wwc)) return FALSE;
  return TRUE;
 } /* CreateClass() */


/*********************************************************************

 —— Instance Dependent Initialization (For Every Instance) ——

*********************************************************************/


/*********************************************************************
 C r e a t e A p p W i n d o w
=====================================================================

All global existing windows are created by this function.

Parameters:
 none

Return:
TRUE is returned if all data read, otherwise FALSE (memory too small).

*********************************************************************/

BOOL CreateAppWindow(VOID)

 {int i;
  BYTE z[50];

  /* create application window */
  if (!CreateWindow
   (zAppName,rzAppTitle,WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN,
    CW_USEDEFAULT,0,CW_USEDEFAULT,0,NULL,NULL,hiMain,NULL)
   )
    return FALSE;
  return TRUE;
 } /* CreateAppWindow() */


/*********************************************************************
 I n i t I n s t a n c e
=====================================================================

### The main init module entry point ###

The complete initialization of a new instance of the application
window. If no previous instance exists, the application window is
initialized and the common data for all instances are created.
Otherwise, common data are copied from the previous instance to the
new instance. All individual data in the instance data segment are
initialized. If memory is too small, a error message box is displayed
and FALSE is returned.

Parameters:
 hiNew .... is the handle to the new instance of the application.
 hiPrev ... is the handle to the first instance of the application
            (or NULL).
 rzCmdLine  points to the command line buffer.
 vCmdShow   is the entry style of window for ShowWindow().

Return:
 TRUE is returned if initialization complete,
 otherwise FALSE (memory too small).

*********************************************************************/

BOOL InitInstance
     (HANDLE hiNew,HANDLE hiPrev,LPSTR rzCmdLine,int vCmdShow)

 {hiMain=hiNew; /* set instance global */
  /* —— load the two strings for the no-memory error message —— */
   {BYTE z[80];
    WORD sz;
    sz=LoadString(hiMain,STAPPTITLE,z,sizeof(z));
    rzAppTitle=(BYTE*)LocalAlloc(LMEM_FIXED,sz+1);
    if (!rzAppTitle) return FALSE;
```

• Bedenken Sie, daß alle Optimierungen überflüssig sind, wenn das Fenster vollständig oder zumindest weiträumig gezeichnet werden muß. Aber die Abfragen, ob bestimmte Objekte im Zeichenbereich liegen oder nicht, brauchen auch dann Zeit. Wenn dann das Neuzeichnen wegen zahlreicher Optimierungen erheblich länger dauert, haben Sie wirklich unnötigen Code erzeugt.

Zunächst ist man bei der Windows-Programmierung sicherlich froh, überhaupt eine gewünschte Ausgabe korrekt auf den Bildschirm gebracht zu haben, so daß man nicht gewillt ist, sich auch noch Gedanken um die Geschwindigkeit seiner »grafischen Kunst« zu machen. Doch mit zunehmender Sicherheit im Umgang mit dem Windows-API wird man auch dem Anwender die Freude bereiten wollen, zumindest bei Routine-Arbeiten wie Blättern oder Rollen eine hohe Geschwindigkeit zu erreichen. Dies wurde auch in der vorgestellten Applikation versucht, obwohl auch hier sicherlich noch viele Verbesserungen möglich sind.

Wie oft bei Windows gehören zur Optimierung der Bildausgabe eine gewisse Erfahrung und ein Gefühl dafür, was bei der Grafikausgabe viel Zeit und was weniger Zeit benötigt. Leider helfen einem hierbei die Windows-SDK-Dokumentation und auch die meisten Windows-Lehrbücher nicht viel weiter.

Schnelle Textausgabe unter Windows

Textausgabe mit mehreren Zeilen innerhalb eines Fensters wird verhältnismäßig oft gebraucht. Außerdem wird sie immer gerne als Vergleich zwischen Windows und DOS herangezogen, wenn über die Ausgabegeschwindigkeit beider Oberflächen diskutiert wird. Inzwischen hat sie bei Windows fast die Geschwindigkeit von DOS erreicht, wenn dort jedes Zeichen einzeln über DOS und ANSI.SYS ausgegeben wird. Sicherlich liegt dies nicht nur an den »genialen« Grafikalgorithmen von Windows 2, sondern auch an dem schlechten Code von DOS und ANSI.SYS. Verglichen mit der Ausgabe von Zeichen direkt in den Bildschirmspeicher ist Windows natürlich weiterhin chancenlos. Dies wird sich wohl erst ändern, wenn in Zukunft Grafikprozessoren auf dem Markt erscheinen werden, die eine Windows-verträgliche Proportionalschrift-Ausgabe in Hardware realisieren können.

Man sollte sich also Gedanken machen, wie man Text unter Windows möglichst schnell »in die Pixel« bringt. Ich habe dies mit einem kleinen Benchmark-Programm gemacht und war über die Ergebnisse, und insbesondere über die weite Spannweite verschiedener Optionen, ziemlich überrascht. Faktoren von 1:7 bei weitgehend gleichen Parametern konnten beobachtet wer-

den. Ich kann im Rahmen dieses Artikels nicht im Detail auf die Messungen eingehen, aber ich plane ein Windows-Grafik-Benchmark-Programm, mit dem man die Ausgabegeschwindigkeit verschiedener GDI-Funktionen messen kann, was insbesondere beim Vergleich von Grafik-Karten für Windows interessant sein dürfte. Vielleicht wird es darüber in absehbarer Zeit im Microsoft System Journal einen Artikel geben.

Doch einige Ergebnisse sollen hier kurz zusammengetragen werden. Zunächst hat mich erstaunt, daß die Funktion `ExtTextOut`, die erst mit Windows 2.0 in den API kam, immer schneller war als `TextOut` (im Extrem doppelt so schnell), sofern beim Aufruf ersterer weder ein zusätzliches Zeichenrechteck noch eine Abstandstabelle angegeben wurde. Da `ExtTextOut` alle wesentlichen Eigenschaften von `TextOut` besitzt, kann man erstere also generell letzterer vorziehen. Weiterhin hat mich bei beiden Funktionen verblüfft, daß die Ausgabe von 30 Zeichen innerhalb des Zeichenausschnitts nur rund dreimal länger dauert als die Ausgabe von drei Zeichen. Es wird also viel Zeit verwendet, die Angaben eines Funktionsaufrufs zu überprüfen und mit dem Zeichenausschnitt abzugleichen. Die eigentliche Ausgabe erfolgt dann mit verhältnismäßig hoher Geschwindigkeit pro Zeichen. Gibt man 30 Zeichen aus, von denen sich aber nur drei Zeichen im Zeichenausschnitt befinden, ist die Ausgabe kaum langsamer, als wenn man von vorne herein nur drei Zeichen ausgegeben hätte. Die Reduktion auf den (im Test rechteckigen) Zeichenausschnitt durch den GDI funktioniert also sehr gut. Ob man die Option `OPAQUE` oder `TRANSPARENT` vorher eingestellt hat, verändert die Ausgabegeschwindigkeit bei drei Zeichen nur unwesentlich, bei 30 Zeichen war allerdings die Geschwindigkeit mit Option `OPAQUE` etwa 20% schneller. Bei Verwendung von `ExtTextOut` mit Angabe einer Abstandstabelle nahm die Ausgabezeit deutlich zu, bei 30 Zeichen war sie etwa dreimal so lang. Trotzdem ist diese Methode etwa bei der Ausgabe von Tabellen mit Proportionalschrift weit der Methode überlegen, jeden Tabelleneintrag einzeln und dann ohne explizite Zeichenabstands-Angaben auszugeben.

Schließlich wurde noch untersucht, wie sich die Startposition der Ausgabe auswirkt. Einmal wurden Zeichen an einer horizontalen Byte-Grenze im Videospeicher ausgegeben, im zweiten Fall an einer Position inmitten eines Bytes (um 4 Bit verschoben). Bei einer Ausgabe von 30 Zeichen war die Ausgabe im ersten Fall mit `ExtTextOut` etwa doppelt so schnell, wobei allerdings angemerkt werden muß, daß alle Zeichen auch acht Pixel breit waren. Die Messungen wurden alle auf einem AT-System mit der VEGA-VGA-Karte von Video7 vorgenommen. Weitere Untersuchungen wurden noch nicht durchgeführt, da die erhaltenen Resultate für die DUMP-Applikation zunächst einmal ausreichten.

◄ Listing 1: (Ende)

```
    strcpy(rzAppTitle,z);
    sz=LoadString(hiMain,STNOMEM,z,sizeof(z));
    rzNoMemory=(BYTE*)LocalAlloc(LMEM_FIXED,sz+1);
    if (!rzNoMemory) return FALSE;
    strcpy(rzNoMemory,z);
   } /* block */
  /* —— initialize first/further instance —— */
  if (!hiPrev)
   {/* first instance: create window class, exit if error */
    if (!CreateClass()) goto MEM_ERROR;
   } /* if */
  InitPaneManager(hiPrev);
  /* create global application window */
  if (!CreateAppWindow()) goto MEM_ERROR;
  /* show created main window */
  ShowWindow(hwMain,vCmdShow); UpdateWindow(hwMain);
  return TRUE;
  /* ===== error: memory too small ===== */
  MEM_ERROR:
  ErrorNoMem(NULL);
  return FALSE; /* return with error */
 } /* InitInstance() */


/*******************************************************************
 ———————————————— Main function ————————————————
*******************************************************************/

WORD PASCAL WinMain
     (HANDLE hiNew,HANDLE hiPrev,LPSTR rzCmdLine,int vCmdShow)

 {MSG wm;

  /* Initialize (window and) instance, return if error */
  if (!InitInstance(hiNew,hiPrev,rzCmdLine,vCmdShow)) return 255;
  /* — application execution loop — */
  while (GetMessage(&wm,NULL,0,0))
   {TranslateMessage(&wm); DispatchMessage(&wm);
   } /* while */
  return wm.wParam;
 } /* WinMain() */


/* ==========================================
   Windows application DUMP: end of module DUMP
   ==========================================
*/
```

Wie wurden nun die Meßergebnisse für DUMP umgesetzt? Zunächst wurde generell die Funktion `ExtTextOut` für die Textausgabe verwendet. Dann wurde beschlossen, immer eine ganze Zeile komplett aufzubauen, unabhängig, welche Teile nun angezeigt oder aktualisiert werden müssen. Dies vereinfacht den Datenaufbau erheblich und führt andererseits laut Meßergebnissen kaum zu Verzögerungen, auch wenn tatsächlich nur wenig Text ausgegeben werden muß. Die Abstände zwischen den einzelnen Zeichen variieren öfters, so wurde etwa zwischen jedem Hex-Byte-Wert nur ein halber Leerraum verwendet, außer nach 8 Bytes, da hier eine Trennlinie eingesetzt wird. Ähnlich verhält es sich bei der Trennung zu der ASCII-Zeichen-Darstellung. Es wurde also für `ExtTextOut` eine Abstandstabelle verwendet: Auch wenn diese die Ausgabe deutlich verlangsamt, so erhöht sie dennoch die Lesbarkeit der Darstellung und außerdem ist die Ausgabe schmäler geworden, da nicht immer volle Leerzeichen eingefügt werden mußten.

Die Ausgabe der vorgestellten Applikation ist auch in einem großen Applikationsfenster ziemlich schnell. Rufen Sie zum Vergleich die Applikation HEAPWALK auf und lassen Sie sich damit in einem gleich großen Fenster zum Vergleich Hex-Daten anzeigen. Der Unterschied bei der Ausgabegeschwindigkeit fällt schon deutlich ins Gewicht, wenn man einmal hin- und herblättert. Und dies, obwohl HEAPWALK die Abstände unergonomisch zwischen den Hex-Werten überall gleichläßt und auch keine vertikalen oder horizontalen Trennlinien zieht und somit die Ausgabe ähnlich langweilig und unübersichtlich aussieht wie bei DOS mit 80x25 Zeichen.

Ihr Know-how ist gefragt!

Sie als Leser des Microsoft System Journals sind Experte auf dem Gebiet der Software, sowohl bei der Programmierung als auch in der Anwendung. An diesem Know-how sind viele andere Programmierer und Anwender brennend interessiert. Der Synergy Verlag sucht deshalb Programmierer, Berater und erfahrene Anwender, die ihr Wissen im Microsoft System Journal publizieren möchten. Wenn Sie also Interesse daran haben, Ihr Know-how zu vermarkten, wenden Sie sich an:

Synergy Verlag GmbH, Hartmut Niemeier, Theresienstr. 40, 8000 München 2, Tel.: 089/280685

► *Listing 2: Das Modul des Pane-Managers, PANE.C.*

```
/*********************************************************************

 —— D U M P ———————— MS-Windows Application ——————————
                      Module PANE.C

=====================================================================

This module contains the complete pane management for the DUMP
application.

_____________________________________________________________________

Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

Release 1.00 of 89-Sep-13 —— All rights reserved.

*********************************************************************/

/* read common header files */
#include "DUMP.H"


/*********************************************************************

 ——————————————— Pane-Management-Variables ———————————————

*********************************************************************/

BOOL bKeySplit; /* the pane size is changing by using the keyboard */

/* split box window style */
#define SPS_HORZ 0 /* horizontal split box */
#define SPS_VERT 1 /* vertical split box */

/* message for split box */
#define SPM_SPLIT  WM_USER /* split command */

/* dialog element sizes */
int sxMullion; /* width of horizontal split area */
int syTransom; /* height of vertical split area */
int syHScroll; /* height of horizontal scroll bar */
int sxVScroll; /* width of vertical scroll bar */

/* pointer to resources */
HCURSOR hcrSplitH; /* horizontal split for panes */
HCURSOR hcrSplitV; /* vertical split for panes */
HCURSOR hcrSplitX; /* full split for panes */


/*********************************************************************
 S e t S c r o l l V a l u e s
=====================================================================

This function calculates and sets the values for all visible scroll
bars. The values are determined by the document size, the current
location and size of the panes. It is assumed that the bottom or right
location of the scroll bar sets the last line or column of the
document to the bottom or right position of the pane.

This function is called by the pane manager if any of the panes
changes its size. It must be called by the application if any of
values in the [rCPS]-structure is changed, except the <.iPos>-values.
The parameter <bRedraw> must be TRUE if this function is called after
changing the size of the document by the application, except the panes
in the pane window are resized afterwards.

Parameters:
 bRedraw  is TRUE if the scroll bars must redrawn and FALSE if not.

Used Globals:
 rCPS

Return:
 none

*********************************************************************/

VOID SetScrollValues(BOOL bRedraw)

 {int i;
  int iMin,iMax,iPos,iExt;
  int nUnits;
  DATASPEC *rSpec;
  SCROLL *rScroll;
  LONG iBase,iNewBase;

  for (i=0,rScroll=rCPS->Scroll;i<4;i++,rScroll++)
   {if (!rScroll->bVisible) continue; /* only visible scroll bars */
    rSpec=i&1? &rCPS->H:&rCPS->V; iBase=rSpec->iBase[i>=2];
    /* determine number of vertical/horizontal device units */
    switch (i)
     {case 0:
        nUnits=rCPS->Pane[0].wrc.bottom-
               rCPS->Pane[0].wrc.top-rSpec->sHead;
        break;
      case 1:
        nUnits=rCPS->Pane[0].wrc.right-
               rCPS->Pane[0].wrc.left-rSpec->sHead;
        break;
      case 2:
        nUnits=rCPS->Pane[2].wrc.bottom-rCPS->Pane[2].wrc.top; break;
      case 3:
        nUnits=rCPS->Pane[1].wrc.right-rCPS->Pane[1].wrc.left; break;
     } /* switch */
    /* ignore median, divide by unit's height/width */
    nUnits=(nUnits+rSpec->sMedian)/rSpec->sUnit;
    if (nUnits<0) nUnits=0; /* minimum must be 0 */
    /* enlarge extend if needed */
    if (rSpec->uExt==0) rSpec->uExt=1;
    iExt=(int)((rSpec->iMax-rSpec->iMin+32767)/32768);
    if (iExt>rSpec->uExt) rSpec->uExt=iExt;
    /* calculate minimum and maximum bar value (round down and up) */
    iMin=(int)(rSpec->iMin/rSpec->uExt);
    iMax=(int)((rSpec->iMax+rSpec->uExt-nUnits)/rSpec->uExt);
    if (iMax<iMin) iMax=iMin; /* set minimum, invalid. scroll bar */
    /* calculate pos. value in bar (round up/down), reduce to range */
    iPos=(int)((iBase+rSpec->uExt/2)/rSpec->uExt);
    if (iPos<iMin) iPos=iMin; else if (iPos>iMax) iPos=iMax;
    if (iBase<(iNewBase=(LONG)iMin*rSpec->uExt)||
        iBase>(iNewBase=(LONG)iMax*rSpec->uExt)
       )
     {/* reduce base to valid area */
      rSpec->iBase[i>=2]=iNewBase;
     } /* if */

    /* —— set scroll bar values —— */
    rScroll->iMin=iMin; rScroll->iMax=iMax; rScroll->iPos=iPos;
    rScroll->nUnits=nUnits; /* base elements per height/width */
```

Noch eine Bemerkung am Rande: Bei einer durchschnittlichen Applikationsfenster-Größe werden etwa 300 bis 400 Hex-Werte beim Neuschreiben einer Bildschirmseite ausgegeben. In einer frühen Version des DUMP-Programms wurde für die Erzeugung der Hex-Werte die Funktion `sprintf` verwendet. Als probehalber statt Hexzeichen einfach konstante Strings ausgegeben wurden (weil ein anderer Fehler partout nicht verschwinden wollte), ergab sich überraschend, daß `sprintf` aufgrund seiner hohen Komplexität bei so vielen Aufrufen ebenfalls ein Hemmschuh für die Ausgabegeschwindigkeit ist. Es wurde daher die Ausgabe der zweistelligen Byte-Hex-Werte »zu Fuß« programmiert, mit dem Erfolg, daß sich die Bildschirm-Ausgabegeschwindigkeit danach fast noch einmal verdoppelt hatte. Manchmal sind eben nicht nur die komplexen Windows-Grafik-Funktionen die Bremser, sondern auch die guten alten C-Funktionen.

Das Ziehen des Rollfelds

Bekanntlich kann mit einer Rolleiste nicht nur gerollt und geblättert werden, sondern das Rollfeld (im Original *thumb box*) kann mit der Maus auch zu einer neuen Position gezogen werden. Dies ermöglichen fast alle Windows-Applikationen als weitere Alternative zum Blättern und Rollen. Üblicherweise wird dabei der Fensterinhalt jedoch nicht sofort mitbewegt. Erst wenn das Ziehen des Rollfelds beendet wird, indem die Maustaste losgelassen wird, erscheint im Fenster *der* Inhalt des Dokuments, der mit der neuen Position korrespondiert. Will man etwa an den Anfang oder das Ende des Dokuments gelangen, ist dies sicherlich praktikabel. Weniger nützlich ist die Methode dagegen, wenn man eine bestimmte Stelle sucht. Man kann das Rollfeld weder bei MS-Write benutzen, um eine bestimmte Seite des Dokuments auszuwählen, noch im Notizblock, um eine Zeile auszuwählen. Bei MS-Excel wurde das Ziehen des Rollfelds dagegen ergonomisch verbessert: Während des Ziehens wird die aktuelle Zeilen- und Spaltenposition angezeigt, die sich auf die linke obere Fensterecke bezieht. So kann man leicht eine gewünschte Position ansteuern, ohne hierfür Menüeinträge oder Dialogfelder zu benötigen.

Beim Hex-Dump wird eine ähnliche Technik verwendet: Während des vertikalen Ziehens mit der Maus wird die erste Adresse im vertikalen Kopf der Anzeige ständig aktualisiert, so daß sie immer mit der gegenwärtigen Ziehposition des Rollfelds übereinstimmt. Zur Verdeutlichung wird dabei der Wert in einem invertierten Hintergrund-Rechteck angezeigt. Erst wenn die Maustaste losgelassen und das Ziehen beendet wird, wird der Fensterinhalt vollständig gezeichnet, wobei in der ersten Zeile immer die Adresse

steht, die bereits vorher während des Ziehens angezeigt wird. Man gewöhnt sich schnell an diese Anzeige und sie stellt eine große Hilfe dar, wenn man sich kurzfristig den Inhalt einer bestimmten Adresse der geladenen Datei ansehen möchte.

Das Ziehen des Rollfelds mit der Maus bei großen Dokumenten hat einen Nachteil: Da eine Rolleiste nur eine begrenzte Auflösung hat, die durch die Anzahl ihrer Bildpunkte in der Länge vorgegeben ist, kann nicht mehr zeilengenau positioniert werden, sobald die Anzahl der Zeilen die der Bildpunkte übersteigt. Ein Dialogfeld zur Eingabe der gewünschten Position ist in solchen Fällen in der Positioniergenauigkeit der Rolleisten-Auswahl überlegen. Man kann aber auch beim Ziehen der Rolleiste zunächst eine naheliegende Position ansteuern und dann mit einigem Vor- oder Zurückblättern an das gewünschte Ziel gelangen. Obwohl diese Methode nicht sehr systematisch ist, dürfte sie für den geübten Benutzer die schnellere sein – vorausgesetzt das Blättern am Bildschirm vollzieht sich ausreichend schnell.

Wenn Sie den Windows-HeapWalker aufrufen, sich ein Hex-Dump ansehen und in diesem durch Ziehen des Rollfelds an eine andere Position gelangen wollen, wird ihnen auffallen, daß dieses Programm bei jeder Zugbewegung sofort den Fensterinhalt komplett aktualisiert. Da dies verhältnismäßig langsam erfolgt und ein Weiterziehen des Rollfelds während des Zeichnens des Inhalts nicht möglich ist, ist bei der Darstellung großer Datenblöcke die Auswahl einer gewissen Adresse ziemlich chaotisch und langwierig. Eine Aktualisierung des kompletten Fensterinhalts während des Ziehens des Rollfelds ist nur dann praktikabel, wenn die Ausgabe sofort unterbrochen wird, sobald das Rollfeld weiterbewegt wird, also das Ziehen in keiner Weise verzögert wird. Dies läßt sich durch den Einbau eines `PeekMessage`-Aufrufs in die Ausgabe-Routine erreichen und wird beispielsweise beim Micrografx-Designer teilweise realisiert. Diese Programmiertechnik führt allerdings zu einer verhältnismäßig unübersichtlichen Ausgabefunktion [5]. Die aktualisierte Anzeige eines festen kleinen Ausschnitts während des Ziehens wie bei der DUMP-Applikationen ist sicherlich einfacher. Die Methode, die beim HeapWalker verwendet wird, ist dagegen nur praktikabel, wenn wenn die Bildschirm-Ausgabe bei MS-Windows um ein vielfaches schneller wird. Und da werden wir wohl noch einige Jahre warten müssen ...

Virtuelle Datenhaltung mittels der Windows-Speicherverwaltung

Die vorgestellte Applikation kann jeden beliebigen Bereich einer Datei mit bis zu 64 Mbyte

```
   SetScrollRange(rScroll->hw,SB_CTL,iMin,iMax,FALSE);
   SetScrollPos(rScroll->hw,SB_CTL,iPos,bRedraw);
  } /* for */
 } /* SetScrollValues() */


/*********************************************************************
 S e t P a n e W i n d o w
=====================================================================

This function sets the interior of the pane window. This function is
called if the size of the pane window is changed or the splitting of
the window context is changed.

Parameters:
 sxWindow  is the (new) width of the window.
 syWindow  is the (new) height of the window.

Used variables:
 rCPS
Return:
 none
*********************************************************************/

VOID SetPaneWindow(int sxWindow,int syWindow)

 {RECT wrcClient;
  struct
   {POINT p;
    int sx,sy;
   }
  ScrollBar[4];
  int i;
  int nCont;

  if (GetFocus()==rCPS->hwPane && rCPS->Pane[0].wrc.left!=-1)
   {/* turn off current focus */
    SetPaneFocus(-1,FALSE);
   } /* if */
  /* —— determine new size of pane client rectangles —— */
  rCPS->Pane[0].wrc.left=0; rCPS->Pane[0].wrc.top=0;
  rCPS->Pane[0].wrc.right=sxWindow-sxVScroll+2;
  rCPS->Pane[0].wrc.bottom=syWindow-syHScroll+2;
  rCPS->Pane[1]=rCPS->Pane[0];
  rCPS->Pane[2]=rCPS->Pane[0]; rCPS->Pane[3]=rCPS->Pane[0];
  rCPS->Pane[0].bValid=TRUE;
  rCPS->Pane[1].bValid=rCPS->Pane[2].bValid=
                       rCPS->Pane[3].bValid=FALSE;
  if (rCPS->pxMullion!=0)
   {/* divide pane 0 and 1 */
    rCPS->Pane[0].wrc.right=rCPS->Pane[0].wrc.left+rCPS->pxMullion;
    rCPS->Pane[1].wrc.left+=rCPS->pxMullion+sxMullion;
    rCPS->Pane[1].bValid=TRUE;
   } /* if */
  if (rCPS->pyTransom!=0)
   {/* divide pane 0 and 2 */
    rCPS->Pane[0].wrc.bottom=rCPS->Pane[0].wrc.top+rCPS->pyTransom;
    rCPS->Pane[2].wrc.top+=rCPS->pyTransom+syTransom;
    rCPS->Pane[2].bValid=TRUE;
    if (rCPS->pxMullion!=0)
     {/* determine part 3 */
      rCPS->Pane[2].wrc.right=rCPS->Pane[0].wrc.right;
      rCPS->Pane[1].wrc.bottom=rCPS->Pane[0].wrc.bottom;
      rCPS->Pane[3].wrc.left=rCPS->Pane[1].wrc.left;
      rCPS->Pane[3].wrc.top=rCPS->Pane[2].wrc.top;
      rCPS->Pane[3].bValid=TRUE;
     } /* if */
   } /* if */
  /* check current selected pane */
  if (!rCPS->Pane[rCPS->iFocusPane].bValid)
   {/* pane with focus doesn't exists: set to pane 0 */
    rCPS->iFocusPane=0;
   } /* if */
  /* —— determine scroll bar locations, sizes and values —— */
  /* vertical (top) scroll bar */
  ScrollBar[0].p.x=rCPS->Pane[1].wrc.right;
  ScrollBar[0].p.y=rCPS->Pane[0].wrc.top+(rCPS->pyTransom?
                  -1:syTransom-1);
  ScrollBar[0].sx=sxVScroll-1;
  ScrollBar[0].sy=rCPS->Pane[0].wrc.bottom-rCPS->Pane[0].wrc.top-
                  (rCPS->pyTransom? -2:syTransom-2);
  /* horizontal (left) scroll bar */
  ScrollBar[1].p.x=rCPS->Pane[0].wrc.left+(rCPS->pxMullion?
                  -1:sxMullion-1);
  ScrollBar[1].p.y=rCPS->Pane[2].wrc.bottom;
  ScrollBar[1].sx=rCPS->Pane[0].wrc.right-rCPS->Pane[0].wrc.left-
                  (rCPS->pxMullion? -2:sxMullion-2);
  ScrollBar[1].sy=syHScroll-1;
  rCPS->Scroll[0].bVisible=rCPS->Scroll[1].bVisible=TRUE;
  if (rCPS->pyTransom)
   {/* set vertical bottom scroll bar */
    rCPS->Scroll[2].bVisible=TRUE;
    ScrollBar[2]=ScrollBar[0];
    ScrollBar[2].p.y=rCPS->Pane[2].wrc.top-1;
    ScrollBar[2].sy=rCPS->Pane[2].wrc.bottom-rCPS->Pane[2].wrc.top+2;
   }
  else
   {/* vertical bottom scroll bar not visible */
    rCPS->Scroll[2].bVisible=FALSE;
   } /* if */
  if (rCPS->pxMullion)
   {/* set horizontal right scroll bar */
    rCPS->Scroll[3].bVisible=TRUE;
    ScrollBar[3]=ScrollBar[1];
    ScrollBar[3].p.x=rCPS->Pane[1].wrc.left-1;
    ScrollBar[3].sx=rCPS->Pane[1].wrc.right-rCPS->Pane[1].wrc.left+2;
   }
  else
   {/* horizontal right scroll bar not visible */
    rCPS->Scroll[3].bVisible=FALSE;
   } /* if */
  SetScrollValues(FALSE); /* set scroll bar ranges & positions */
  /* —— set split fields —— */
  SetWindowPos
   (rCPS->hwHSplit,NULL,rCPS->Pane[0].wrc.left+rCPS->pxMullion,
    rCPS->pxMullion? rCPS->Pane[0].wrc.top:rCPS->Pane[2].wrc.bottom,
    sxMullion,
    rCPS->pxMullion? syWindow-rCPS->Pane[0].wrc.top+1:syHScroll-1,
    SWP_NOACTIVATE|SWP_NOZORDER|SWP_NOREDRAW
   );
  InvalidateRect(rCPS->hwHSplit,NULL,TRUE);
  if (!IsWindowVisible(rCPS->hwHSplit))
    ShowWindow(rCPS->hwHSplit,SW_SHOWNA);
  SetWindowPos
   (rCPS->hwVSplit,NULL,
    rCPS->pyTransom? rCPS->Pane[0].wrc.left:rCPS->Pane[1].wrc.right,
    rCPS->Pane[0].wrc.top+rCPS->pyTransom,
    rCPS->pyTransom? sxWindow-rCPS->Pane[0].wrc.left+1:sxVScroll-1,
    syTransom,
    SWP_NOACTIVATE|SWP_NOZORDER|SWP_NOREDRAW
   );
```

◄ *Listing 2: (Fortsetzung)*

► *Listing 2: (Fortsetzung)*

```
InvalidateRect(rCPS->hwVSplit,NULL,TRUE);
if (!IsWindowVisible(rCPS->hwVSplit))
   ShowWindow(rCPS->hwVSplit,SW_SHOWNA);
/* ——— repositionate scroll bars ——— */
for (i=0;i<4;i++)
 {/* draw scroll bar ScrollBar[i] if exists */
  if (rCPS->Scroll[i].bVisible)
   {/* draw scroll bar */
    SetWindowPos
     (rCPS->Scroll[i].hw,NULL,ScrollBar[i].p.x,ScrollBar[i].p.y,
      ScrollBar[i].sx,ScrollBar[i].sy,
      SWP_NOACTIVATE|SWP_NOZORDER|SWP_NOREDRAW
     );
    InvalidateRect(rCPS->Scroll[i].hw,NULL,TRUE);
    /* show scroll bar if it is not visible */
    if (!IsWindowVisible(rCPS->Scroll[i].hw))
      ShowWindow(rCPS->Scroll[i].hw,SW_SHOWNA);
   }
  else
   {/* scroll bar is not visible: hide it if needed */
    if (IsWindowVisible(rCPS->Scroll[i].hw))
      ShowWindow(rCPS->Scroll[i].hw,SW_HIDE);
   } /* if */
 } /* for */
/* ——— redraw pane window ——— */
InvalidateRect(rCPS->hwPane,NULL,TRUE); UpdateWindow(rCPS->hwPane);
if (GetFocus()==rCPS->hwPane)
 {/* turn on new/current focus */
  SetPaneFocus(rCPS->iFocusPane,FALSE);
 } /* if */
} /* SetPaneWindow() */


/*********************************************************************
 S c r o l l P a n e
=====================================================================

This function analyses the scroll command and scrolls the relevant
panes. The scroll command can be entered via the keyboard or a scroll
bar.

Parameters:
 iScroll  is the index of the scroll bar (0..3) or -1 if the scroll
          command was entered via keyboard. In the last case the
          activate pane and its connected pane is scrolled.
 uCmd ... is the scroll command code. Following values are possible:
            SB_TOP ........... to first line or first column
            SB_BOTTOM ........ to last line or last column
            SB_LINEUP ........ one line up or one column left
            SB_LINEDOWN ...... one line down or one column right
            SB_PAGEUP ........ one page up or one page left
            SB_PAGEDOWN ...... one page down or one page right
            SB_THUMBPOSITION   set to position <iNewPos>
            SB_THUMBTRACK .... drag to position <iNewPos>
 iNewPos  is only active if <uCmd> is SB_THUMBPOSITION or
          SB_THUMBTRACK. It contains the set value of the scroll bar.

Return:
 none

*********************************************************************/

VOID ScrollPane(int iScroll,WORD uCmd,int iNewPos)

 {LONG iBase;
  LONG *riBase;
  int  iSbPos;
  SCROLL *rScroll;
  DATASPEC *rSpec;
  int nUnitsPerPage;

  /* set pointers and index to data structures */
  rScroll=rCPS->Scroll+iScroll; rSpec=iScroll&1? &rCPS->H:&rCPS->V;
  riBase=rSpec->iBase+(iScroll>=2); iBase=*riBase;
  /* reduce units per page if page contains more than one unit */
  nUnitsPerPage=rScroll->nUnits-(rScroll->nUnits>1);
  switch (uCmd)
   {case SB_TOP:
      iBase=rSpec->iMin; break;
    case SB_BOTTOM:
      iBase=max(rSpec->iMin,rSpec->iMax-nUnitsPerPage); break;
    case SB_LINEUP:
      if (iBase>rSpec->iMin) iBase--;
      break;
    case SB_LINEDOWN:
      if (iBase<rSpec->iMax-nUnitsPerPage) iBase++;
      break;
    case SB_PAGEUP:
      iBase=max(iBase-nUnitsPerPage,rSpec->iMin); break;
    case SB_PAGEDOWN:
      if (rSpec->iMax>nUnitsPerPage)
       {iBase=min(iBase+nUnitsPerPage,rSpec->iMax-nUnitsPerPage);
       } /* if */
      break;
    case SB_THUMBPOSITION:
      if (rCPS->wrcTrackRedraw.left<rCPS->wrcTrackRedraw.right)
       {/* erase track redraw position */
        InvalidateRect(rCPS->hwPane,&rCPS->wrcTrackRedraw,TRUE);
        /* set rectangle to "empty" (invalid) */
        rCPS->wrcTrackRedraw.left=rCPS->wrcTrackRedraw.right;
        UpdateWindow(rCPS->hwPane); /* redraw before scrolling */
       } /* if */
    case SB_THUMBTRACK:
      iBase=(LONG)iNewPos*rSpec->uExt; /* multiply pos. with extent */
      if (iBase<rSpec->iMin) iBase=rSpec->iMin;
        else if (iBase>rSpec->iMax) iBase=rSpec->iMax;
      if (uCmd==SB_THUMBTRACK && !(iScroll&1))
        DrawTrackPos(iScroll,iBase);
   } /* switch */
  if (iBase==*riBase) return; /* no location change */
  /* ——— execute scrolling: update contents of panes ——— */
  SetPaneFocus(-1,FALSE); /* switch off caret */
  if (uCmd!=SB_THUMBTRACK)
   {/* ——— set new scroll bar value ——— */
    RECT wrcScroll;
    iSbPos=(int)((iBase+rSpec->uExt/2)/rSpec->uExt);
    if (iSbPos<rScroll->iMin) iSbPos=rScroll->iMin;
      else if (iSbPos>rScroll->iMax) iSbPos=rScroll->iMax;
    if (iSbPos!=rScroll->iPos)
     {/* set new scroll bar location */
      rScroll->iPos=iSbPos;
      SetScrollPos(rScroll->hw,SB_CTL,iSbPos,TRUE);
     } /* if */

    /* —— determine rectangle to scroll/redraw in pane window —— */
    wrcScroll.left=iScroll==3? rCPS->Pane[1].wrc.left:
      iScroll==1? rCPS->Pane[0].wrc.left+rCPS->H.sHead:
                  rCPS->Pane[0].wrc.left;
    wrcScroll.top=iScroll==2? rCPS->Pane[2].wrc.top:
      iScroll==0? rCPS->Pane[0].wrc.top+rCPS->V.sHead:
                  rCPS->Pane[0].wrc.top;
```

Länge anzeigen. Nun gibt es bis heute keine PCs mit 64 Mbyte RAM – es kann also immer nur ein Teil der Datei im RAM stehen. Viele Hex-Dump-Programme realisieren dies so, daß in den Speicher immer ein zusammenhängender Ausschnitt der Datei geladen wird, wobei sich in diesem Ausschnitt auch die aktuell angezeigte Position befindet. Dies ist aber ungünstig, wenn man innerhalb der Datei öfters hin- und herspringen muß, etwa weil man Adressen im Dateikopf gelesen hat und sich anschließend die Daten an diesen Adressen ansehen möchte. In diesem Fall muß ständig auf die Platte zugegriffen werden und mehr oder weniger große Bereiche nachgeladen werden. Es ist also sinnvoll, mehrere getrennte Dateiausschnitte im Speicher zu halten, sofern dieser es von der Größe her erlaubt. Nur dann, wenn sich anzuzeigende Daten noch nicht im Speicher befinden, müssen sie (zeitaufwendiger) von der Datei nachgeladen werden. Irgendwann ist natürlich bei größeren Dateien der RAM-Speicher erschöpft und dann müssen Ausschnitte wieder im Speicher gelöscht werden, um anderen Platz zu machen. Im allgemeinen werden hierzu Ausschnitte verwendet, auf die schon lange nicht mehr zugegriffen wurde. Diese Auslagerungsstategie wird mit *LRU-(least recently used)*-Technik bezeichnet. Sinnvollerweise besitzen die einzelnen Ausschnitte gleiche Größe und befinden sich an festen Anfangsadressen der Datei – die Ausschnitte werden dann Blöcke genannt und eine Datei besteht aus einer ununterbrochenen Anordnung von Blöcken. Allgemein wird das Prinzip *virtuelle Speicherverwaltung* genannt. Weitere Informationen darüber können Sie aus [6] und [7] entnehmen.

Will man bei DOS eine solche Speicherverwaltung implementieren, entsteht ein ziemlicher Aufwand. Man greift in der Praxis sinnvollerweise auf Bibliotheken zurück, die solche Verwaltungen unterstützen (etwa [7]). Bei Windows ist dies dagegen ganz einfach: Windows besitzt bereits eine virtuelle Speicherverwaltung in seiner globalen Halde, die unter anderem für Code, Applikations-Datensegmente und Ressourcen verwendet wird. In dieser Halde können mit der Windows-Funktion `GlobalAlloc` weitere Blöcke angelegt werden, die auch über 64 Kbyte Länge hinausgehen können, sofern der Speicherplatz insgesamt ausreicht. Im allgemeinen werden die Blöcke mit der Option `GMEM_MOVEABLE` als verschiebbar gekennzeichnet, so daß sie sich an unterschiedlichen Adressen befinden können. Die Adresse der gespeicherten Daten enthält man mit der Windows-Funktion `GlobalLock`, die gleichzeitig dafür sorgt, daß der Speicherblock von der Speicherverwaltung solange nicht verschoben wird, bis wieder `GlobalUnlock` aufgerufen worden ist. Solche Blöcke werden aber nie vom Windows-System selbständig gelöscht oder auf Platte ausgelagert. Man kann durch übermäßiges Anlegen solcher Blöcke schnell erreichen, daß

dem System kein Speicher mehr zur Verfügung steht und im Prinzip Windows-weit »nichts mehr geht«.

Für die oben geschriebene Speicherverwaltung müssen die Blöcke aber jederzeit aus der globalen Halde vom Windows-System entfernbar sein. Stellen Sie sich vor, Sie haben eine Zeitlang mit DUMP gearbeitet und sich eine Datei mit 200 Kbyte Länge komplett angesehen. Es kann gut sein, daß alle Daten im Speicher Platz gefunden haben und nun dem Windows-System 200 Kbyte weniger Daten zur Verfügung stehen. Wenn Sie jetzt von DUMP zur MSDOS-Applikation wechseln und MS-EXCEL aufrufen, wird letzteres mit Sicherheit die 200 Kbyte Platz benötigen. Wer teilt jetzt aber DUMP mit, daß es seine Blöcke freigeben muß?

Bei Windows wurde das Problem so gelöst, daß man Datenblöcke in der globalen Halde mit der Option GMEM_DISCARDABLE kennzeichnen kann. Solche Blöcke werden dann von der zentralen Speicherverwaltung des Windows-Systems einfach gelöscht, wenn irgendwo zu wenig Speicher vorhanden ist und der LRU-Algorithmus feststellte, daß der Block lange nicht mehr verwendet wurde. In der Applikation, die den Block geladen hatte, wird das Löschen des Blocks dadurch erkannt, daß die Funktion `GlobalLock` beim Aufruf als Adresse 0 zurückgibt. In diesem Fall muß der Block noch einmal explizit mit `GlobalFree` gelöscht, wieder neu angelegt und seine Daten erneut von der Datei geladen werden.

Für DUMP ist diese Verwaltung optimal, da DUMP keine Dateidaten ändern muß. Als einheitliche Blockgröße wurde 4 Kbyte festgelegt. Beim Laden einer Datei bestimmt die Funktion `LoadFile` im Modul `FILE` aus der Dateilänge die Anzahl der erforderlichen Blöcke und legt einen Kopfblock an (der natürlich nicht entfernbar ist), in dem die entsprechenden Bezüge (*Handles*) der einzelnen Blöcke abgelegt sind beziehungsweise NULL, wenn ein Block bisher noch nicht geladen worden war. Die Funktion `ReadDataBlock` ermittelt dann aus der angegebenen Adresse die entsprechende Blocknummer, prüft nach, ob der Block bereits geladen ist und lädt die Daten notfalls von der Datei. Anschließend wird der Bezug des Blocks mit den gültigen Daten zurückgegeben. Es muß aber noch festgestellt werden, ob vielleicht die Daten eines gültigen Bezugs vom Windows-System aus dem Speicher entfernt wurden. Dies erfolgt durch Aufruf der Funktion `GlobalFlags` und der Analyse des Flags `GMEM_DISCARDED`. Ist es `TRUE`, wurden die Daten gelöscht und müssen neu eingelesen werden. Die Funktion `ReadDataBlock` wird direkt aus `DrawPane` aufgerufen, sobald die Daten für die Ausgabe benötigt werden.

Eine ähnliche Technik existiert auch für Blöcke, die Rasterbilder (*Bitmaps*) enthalten. Wird ein solches Bild mit der Funktion `CreateDiscardableBitmap` erzeugt, kann es ebenfalls

◄ *Listing 2: (Fortsetzung)*

```
    wrcScroll.right=iScroll==1? rCPS->Pane[0].wrc.right:
                                rCPS->Pane[1].wrc.right;
    wrcScroll.bottom=iScroll==0? rCPS->Pane[0].wrc.bottom:
                                 rCPS->Pane[2].wrc.bottom;
    if (labs(*riBase-iBase)<(LONG)rScroll->nUnits)
     {/* scroll pane(s) into specified direction, redraw new parts */
      HDC hdc;
      RECT wrcUpd;
      int sxScroll,syScroll;
      int vDelta;
      /* —— calculate scrolling rectangle —— */
      /* determine scrolling offset (negative if scroll down/right */
      vDelta=(int)(*riBase-iBase)*rSpec->sUnit;
      /* —— calculate scrolling range and offsets —— */
      if (iScroll&1)
       {/* horizontal scrolling */
        sxScroll=vDelta; syScroll=0;
        if (vDelta<0)
         {/* move remainder to left */
          wrcScroll.left-=vDelta;
         }
        else
         {/* scroll remainder to right */
          wrcScroll.right-=vDelta;
         } /* if */
       }
      else
       {/* vertical scrolling */
        sxScroll=0; syScroll=vDelta;
        if (vDelta<0)
         {/* move remainder to top */
          wrcScroll.top-=vDelta;
         }
        else
         {/* move remainder to bottom */
          wrcScroll.bottom-=vDelta;
         } /* if */
       } /* if */
      hdc=GetDC(rCPS->hwPane);
      /* execute scrolling, enlarge update rectangle if needed */
      ScrollDC(hdc,sxScroll,syScroll,&wrcScroll,NULL,NULL,&wrcUpd);
      if (sxScroll<0 && wrcUpd.left>wrcUpd.right+sxScroll)
       {wrcUpd.left=wrcUpd.right+sxScroll;
       }
      else if (sxScroll>0 && wrcUpd.right<wrcUpd.left+sxScroll)
       {wrcUpd.right=wrcUpd.left+sxScroll;
       } /* if */
      if (syScroll<0 && wrcUpd.top>wrcUpd.bottom+syScroll)
       {wrcUpd.top=wrcUpd.bottom+syScroll;
       }
      else if (syScroll>0 && wrcUpd.bottom<wrcUpd.top+syScroll)
       {wrcUpd.bottom=wrcUpd.top+syScroll;
       } /* if */
      ReleaseDC(rCPS->hwPane,hdc);
      InvalidateRect(rCPS->hwPane,&wrcUpd,TRUE);
     }
    else
     {/* ——— redraw concerned panes completely new ——— */
      InvalidateRect(rCPS->hwPane,&wrcScroll,TRUE);
     } /* if */
    *riBase=iBase; /* set new location */
    UpdateWindow(rCPS->hwPane); /* draw immediately */
    SetPaneFocus(rCPS->iFocusPane,FALSE); /* turn on active caret */
   } /* if */
 } /* ScrollPane() */


/*****************************************************************

 ———————————————— Pane-Split-Management ————————————————

*****************************************************************/

/*****************************************************************
 G e t C r o s s S p l i t R e c t
=================================================================

This function calculates the rectangle of the intersection of the two
current set split bars. The coordinates of the rectangle are relative
in the window <hw> and stored into <wrcDest>. The function returns
TRUE if the rectangle exists and FALSE if not.

Parameters:
 hw  is the handle of one of split windows.

Used variables:
 rCPS

Return:
 TRUE if the intersection rectangle exists, FALSE if not.

*****************************************************************/

BOOL GetCrossSplitRect(HWND hw,RECT *rwrcDest)

 {RECT wrc1,wrc2;
  POINT p1,p2;
  /* no intersection => return with FALSE */
  if (rCPS->pxMullion==0||rCPS->pyTransom==0) return FALSE;
  GetWindowRect(rCPS->hwHSplit,&wrc1);
  GetWindowRect(rCPS->hwVSplit,&wrc2);
  if (!IntersectRect(&wrc1,&wrc1,&wrc2)) return FALSE;
  p1.x=wrc1.left; p1.y=wrc1.top; p2.x=wrc1.right; p2.y=wrc1.bottom;
  ScreenToClient(hw,&p1); ScreenToClient(hw,&p2);
  SetRect(rwrcDest,p1.x,p1.y,p2.x,p2.y); return TRUE;
 } /* GetCrossSplitRect() */


/*****************************************************************
 G e t C e n t e r e d M u l l i o n
=================================================================

This function returns a centered mullion location of the pane window
with dimensions <wrcPane>.

Parameters:
 wrcPane  is client rectangle of the pane window.

Used variables:
 rwrc  is a pointer to the client rectangle of the pane window.

Return:
 The vertical mullion value is returned.

*****************************************************************/

int GetCenteredMullion(RECT *rwrc)

 {return ((rwrc->right-sxMullion-sxVScroll-rCPS->H.sHead)/
    (2*rCPS->H.sUnit))*rCPS->H.sUnit+rCPS->H.sHead-rCPS->H.sMedian;
 } /* GetCenteredMullion() */
```

▶ *Listing 2: (Fortsetzung)*

```
/**********************************************************************
 G e t C e n t e r e d T r a n s o m
======================================================================

This function returns a centered transom location of the pane window
with dimensions <wrcPane>.

Parameters:
 rwrc  is a pointer to the client rectangle of the pane window.

Used variables:
 rCPS

Return:
 The horizontal transom value is returned.

**********************************************************************/

int GetCenteredTransom(RECT *rwrc)

 {return ((rwrc->bottom-syTransom-syHScroll-rCPS->V.sHead)/
    (2*rCPS->V.sUnit))*rCPS->V.sUnit+rCPS->V.sHead-rCPS->V.sMedian;
 } /* GetCenteredTransom() */


/**********************************************************************
 L i m i t S p l i t P o s
======================================================================

This function checks if the point [rp] is within the pane window
client area which is specified in <wrc>. If so the position is not
changed. Otherwise the location is reduced to the limit values.

Parameters:
 rwrc  is a pointer to the client rectangle of the pane window.
 rp .. is a pointer to the position value.

Used variables:
 rCPS

Return:
 The horizontal transom value is returned.

**********************************************************************/

VOID LimitSplitPos(RECT *rwrc,POINT *rp)

 {int sMax;
  if (rp->x<0) rp->x=0;
    else if (rp->x>(sMax=rwrc->right-sxVScroll-sxMullion+2))
      rp->x=sMax;
  if (rp->y<0) rp->y=0;
    else if (rp->y>(sMax=rwrc->bottom-syHScroll-syTransom+2))
      rp->y=sMax;
 } /* LimitSplitPos() */


/**********************************************************************

 ——————————————— "Split" Window function ———————————————

**********************************************************************/

LONG FAR PASCAL fwSplit(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {BOOL bVert; /* TRUE if vertical splitting */
  static BOOL bSetSplit; /* TRUE if split setting terminated */
  static POINT pSplit; /* current split point (cross of corner) */
  static int sxMouseOffs; /* x-offset between bar and mouse loc. */
  static int syMouseOffs; /* y-offset between bar and mouse loc. */

  bVert=(BOOL)GetWindowLong(hw,GWL_STYLE)&1;
  switch (iMsg)
   {case WM_PAINT:
     {PAINTSTRUCT wps;
      HDC hdc;
      HBRUSH hbrDraw;
      HPEN hpnDraw;
      RECT wrc;
      hdc=BeginPaint(hw,&wps);
      hpnDraw=CreatePen(PS_SOLID,1,GetSysColor(COLOR_WINDOWFRAME));
      hbrDraw=CreateSolidBrush(GetSysColor(COLOR_WINDOWFRAME));
      SelectObject(hdc,hpnDraw);
      if (GetCrossSplitRect(hw,&wrc))
       {/* exclude cross rectangle from clipping area */
        InflateRect(&wrc,bVert? -1:0,bVert? 0:-1);
        ExcludeClipRect(hdc,wrc.left,wrc.top,wrc.right,wrc.bottom);
       } /* if */
      GetClientRect(hw,&wrc);
      if (bVert)
       {if (rCPS->pyTransom!=0)
         {/* draw rectangle of vertical split bar */
          Rectangle(hdc,-1,0,wrc.right-sxVScroll+2,wrc.bottom);
         } /* if */
        /* draw vertical split box */
        SelectObject(hdc,hbrDraw);
        Rectangle(hdc,wrc.right-sxVScroll+1,0,wrc.right-1,wrc.bottom);
       }
      else
       {if (rCPS->pxMullion!=0)
         {/* draw rectangle of horizontal split bar */
          Rectangle(hdc,0,-1,wrc.right,wrc.bottom-syHScroll+2);
         } /* if */
        /* draw horizontal split box */
        SelectObject(hdc,hbrDraw);
        Rectangle
         (hdc,0,wrc.bottom-syHScroll+1,wrc.right,wrc.bottom-1);
       } /* if */
      EndPaint(hw,&wps);
      DeleteObject(hpnDraw); DeleteObject(hbrDraw); return 0L;
     } /* case */
    case WM_LBUTTONDOWN:
      /* during keyboard-using: convert to "release-button" */
      if (bKeySplit) iMsg=WM_LBUTTONUP;
    case WM_MOUSEMOVE:
    case WM_LBUTTONDBLCLK:
    case WM_LBUTTONUP:
    case SPM_SPLIT:
     {RECT wrcCross; /* intersection rect. of the both split bars */
      POINT pMouse; /* current mouse location */
      BOOL bCross; /* TRUE if mouse location within split bar cross */
      static BOOL bSplitting; /* during splitting operation */
      static BOOL bHSet; /* horizontal setting of mullion */
      static BOOL bVSet; /* vertical setting of transom */
      HWND hwPane=GetParent(hw); /* handle of pane-window */
      if (iMsg==SPM_SPLIT)
       {/* —— start splitting with keyboard-/mouse-interface —— */
        RECT wrc;
        POINT pMouseSet;
        GetClientRect(hwPane,&wrc); /* client of parent window */
        bCross=TRUE; bHSet=TRUE; bVSet=TRUE;
```

vom System aus dem Speicher entfernt werden. Wird auf das Bild oft zugegriffen, befindet es sich fertig aufgebaut im Speicher und kann schnell angezeigt werden. Wurde dagegen das Bild längere Zeit nicht benutzt und wurde der von ihm belegte Speicher für andere Daten dringender benötigt, muß das Rasterbild durch Zeichenoperationen wieder neu aufgebaut werden, wofür dann die Anzeige kurzfristig etwas länger dauert als wenn es schon fertig im Speicher steht. Die Windows-Fensterverwaltung nutzt diese Technik, wenn Menüs aufgeklappt werden, um den darunterliegenden Fensterbereich zu retten. Ist der Speicher groß genug, wird der Bereich als Rasterbild gerettet und das Neuzeichnen nach Schließen des Menüs erfolgt sehr schnell. Ist dagegen der Speicher zu klein, kann der Fensterbereich nicht gerettet werden und die Fensterverwaltung sendet der entsprechenden Fensterfunktion eine `WM_PAINT`-Nachricht, damit der Bereich neu gezeichnet wird, was unter Umständen sehr viel länger dauern kann.

Was macht man nun, wenn in Speicherblöcken Daten stehen, die vom Windows-System nicht einfach gelöscht werden dürfen, etwa weil der Benutzer sie verändert hat und eine Abspeicherung auf Platte erfolgen muß? Bei den meisten modernen Betriebssystemen wie OS/2 werden solche Datenblöcke automatisch und unsichtbar für die Applikationen in eine spezielle »Swap«-Systemdatei auf der Platte ausgelagert und gehen somit nicht verloren. Bei Windows wurde dies bis heute leider aus mir unerfindlichen Gründen nicht realisiert (wahrscheinlich gab es Synchronisationsprobleme mit den DOS-Dateifunktionen).

Ab Windows 2.0 steht ein anderes Konzept zur Verfügung. Es ist zwar umständlicher zu benutzen, aber immerhin in der Praxis einigermaßen brauchbar. Man kann der Windows-Speicherverwaltung mit Hilfe der Funktion `GlobalNotify` die Adresse einer Meldefunktion übergeben, die jedesmal aufgerufen wird, wenn ein Block gelöscht werden soll, der zur gleichen Task wie die Funktion gehört. Die Meldefunktion kann dann veranlassen, daß die Daten in diesem Block vorher auf Platte gesichert werden – diese Auslagerung muß die Applikation also selbst übernehmen. Das hört sich ganz gut an – in der Theorie. In der Praxis wurde erst einmal vergessen, die Funktion `GlobalNotify` in die Dokumentation des Windows-SDK 2 aufzunehmen und wird erst im Update zum SDK beschrieben. Dort wird sie dann mehr oder weniger falsch beschrieben. Tatsache ist: Die angegebene Meldefunktion kann jederzeit vom Windows-System aufgerufen werden, auch wenn die entsprechende Applikation nicht aktiv ist. Sie besitzt ähnliche Eigenschaften wie Systemeingriffe (*Hooks*). Insbesondere sind Besonderheiten im Zusammenhang mit der EMS-Verwaltung zu beachten. Die aufgerufene Meldefunktion muß sich daher in einem

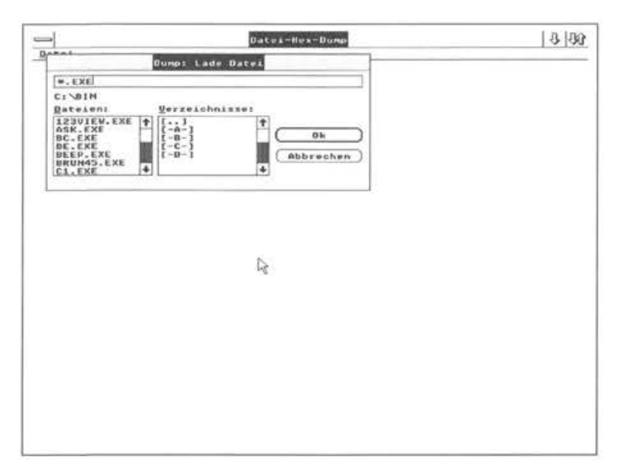


Bild 8: Das Dialogfeld zum Laden von Dateien.

FIXED-Codebereich innerhalb einer Laufzeitbibliothek (DLL) befinden [8]. Innerhalb der Meldefunktion können Stack- und Datensegment unterschiedlich sein. Als Argument wird auch nicht unbedingt der Bezug des Blocks übergeben, der freigegeben werden soll, sondern vielmehr die physikalische Segmentadresse. Der Bezug kann aus letzterer allerdings mit Hilfe der Windows-Funktion `GlobalHandle` ermittelt werden. Die aufgerufene Meldefunktion darf keinesfalls Aktionen ausführen, die zum Freigeben anderer Speicherblöcke führen könnten, also etwa neue globale Speicherblöcke anlegen, den Speicher kompaktieren oder vorhandene Blöcke vergrößern.

Als Beleg dafür, daß die Funktion `GlobalNotify` auch von Microsoft wohl noch nicht allzuoft verwendet worden ist, kann ihre Definition in WINDOWS.H gelten. Statt des erwarteten Argumenttyps FARPROC (für die Adresse der Meldefunktion) wurde versehentlich LPSTR als Typ angegeben, und dies paßt nun wirklich nicht dahin.

Das Datei-Modul

Der komplette Dateizugriff wurde aus didaktischen Gründen in ein extra Modul mit Namen `FILE.C` verlagert. In diesem Modul befindet sich auch die Dialogfeld-Funktion zur Darstellung und Bedienung des Dialogfelds für das Eröffnen einer Datei. Diese Funktion sowie weitere Funktionen, die von ihr aufgerufen werden, werden in ähnlicher Form in fast jede Windows-Applikation integriert. Im vorliegenden Beispiel wurden wie bei MS-Excel die Dateien und Unterverzeichnisse des angewählten Verzeichnisses in unterschiedlichen Listenfeldern angezeigt, siehe *Bild 8*. Dies erhöht einerseits den Überblick über ein Verzeichnis und erlaubt andererseits dem Benutzer den schnelleren Zugriff auf eine gewünschte Datei, da er dafür nicht erst alle Verzeichnisnamen überspringen muß, wie dies der Fall wäre, wenn alle Namen in einem Listenfeld stehen. Die verwendeten Funktionen für das Eröffnen einer Datei sind sonst zum größten Teil Modifikationen aus dem *MS-Windows-Learning-*

Listing 2: (Fortsetzung)

```
    /* set offset between split bars and mouse point */
    sxMouseOffs=sxMullion/2; syMouseOffs=syTransom/2;
    if (rCPS->pxMullion==0 && rCPS->pyTransom==0)
     {/* set start location to centering split */
      pMouse.x=GetCenteredMullion(&wrc);
      pMouse.y=GetCenteredTransom(&wrc);
     }
    else
     {/* start at current setting */
      pMouse.x=0; pMouse.y=0;
      if (rCPS->pxMullion!=0) pMouse.x=rCPS->pxMullion;
      if (rCPS->pyTransom!=0) pMouse.y=rCPS->pyTransom;
     } /* if */
    pMouse.x+=sxMouseOffs; pMouse.y+=syMouseOffs;
    /* set cursor to split location */
    pMouseSet=pMouse;
    ClientToScreen(hwPane,&pMouseSet);
    SetCursor(hcrSplitX); SetCursorPos(pMouseSet.x,pMouseSet.y);
    SetCapture(hw); bSplitting=TRUE; bSetSplit=FALSE;
   }
  else
   {/* —— determine if mouse is within split bar cross —— */
    pMouse.x=P2LO; pMouse.y=P2HI;
    bCross=GetCrossSplitRect(hw,&wrcCross)&&
          PtInRect(&wrcCross,pMouse);
    /* get current mouse position as parent window coordinates */
    ClientToScreen(hw,&pMouse); ScreenToClient(hwPane,&pMouse);
    if (iMsg==WM_MOUSEMOVE||iMsg==WM_LBUTTONDOWN)
     {/* set cursor of field */
      SetCursor(bHSet&&bVSet||
               bCross? hcrSplitX:bVert? hcrSplitV:hcrSplitH);
     } /* if */
   } /* if */
  if (iMsg==WM_LBUTTONDOWN||iMsg==WM_LBUTTONDBLCLK)
   {/* —— enter splitting mode —— */
    SetCapture(hw); bSplitting=TRUE; bSetSplit=FALSE;
    /* set splitting mode */
    if (bCross)
     {/* simultanous change of mullion and transom */
      bHSet=TRUE; bVSet=TRUE;
     }
    else
     {/* change only one direction */
      bHSet=!bVert; bVSet=bVert;
     } /* if */
    /* set mouse position deltas */
    sxMouseOffs=pMouse.x-rCPS->pxMullion;
    syMouseOffs=pMouse.y-rCPS->pyTransom;
   } /* if */
  if (bSplitting)
   {/* ====== redraw splitting line if needed ====== */
    HDC hdc;
    RECT wrc;
    BOOL bVChange=TRUE;
    BOOL bHChange=TRUE;
    GetClientRect(hwPane,&wrc); /* client of parent window */
    /* get and analyse mouse location */
    if (iMsg==WM_LBUTTONDBLCLK)
     {/* —— center split line or destroy it —— */
      if (bHSet)
       {/* analyse mullion: valid => destroy, center otherwise */
        pMouse.x=rCPS->pxMullion? 0:GetCenteredMullion(&wrc);
       } /* if */
      if (bVSet)
       {/* analyse transom: valid => destroy, center otherwise */
        pMouse.y=rCPS->pyTransom? 0:GetCenteredTransom(&wrc);
       } /* if */
      bSetSplit=TRUE; /* end of repositioning */
     }
    else
     {/* determine new mouse location */
      pMouse.x-=sxMouseOffs; pMouse.y-=syMouseOffs;
     } /* if */
    LimitSplitPos(&wrc,&pMouse);
    if (iMsg==WM_LBUTTONUP) bSetSplit=TRUE; /* end of repos. */
    hdc=GetDC(hwPane);
    SelectObject(hdc,GetStockObject(GRAY_BRUSH));
    /* repaint mullion or transom line */
    if (iMsg!=WM_LBUTTONDOWN && iMsg!=WM_LBUTTONDBLCLK &&
        iMsg!=SPM_SPLIT)
     {/* remove old mullion or transom line if needed */
      bHChange=bHSet&&pMouse.x!=pSplit.x;
      bVChange=bVSet&&pMouse.y!=pSplit.y;
      if (bHSet && (bHChange||bSetSplit))
       {/* remove old mullion line marker */
        PatBlt(hdc,pSplit.x,0,sxMullion,wrc.bottom,PATINVERT);
       } /* if */
      if (bVSet && (bVChange||bSetSplit))
       {/* remove old transom line marker */
        PatBlt(hdc,0,pSplit.y,wrc.right,syTransom,PATINVERT);
       } /* if */
     } /* if */
    if (bSetSplit)
     {/* ====== terminate splitting mode ====== */
      if (bHSet&&pMouse.x!=rCPS->pxMullion||
          bVSet&&pMouse.y!=rCPS->pyTransom)
       {/* reorder interior of main application window */
        if (bHSet)
         {/* set new mullion: check width of panes */
          if (pMouse.x<rCPS->H.sHead+
              rCPS->H.sUnit-rCPS->H.sMedian)
           {/* left pane too small: destroy mullion, use right */
            rCPS->H.iBase[0]=rCPS->H.iBase[1];
            pMouse.x=0;
           }
          else if
           (pMouse.x>=
            wrc.right-sxVScroll-rCPS->H.sUnit+rCPS->H.sMedian)
           {/* right pane too small: destroy mullion, use left */
            pMouse.x=0;
           }
          else if (rCPS->pxMullion==0)
           {/* new pane created: set index of base unit in pane */
            rCPS->H.iBase[1]=rCPS->H.iBase[0]+(pMouse.x-
                  rCPS->H.sHead+rCPS->H.sMedian)/rCPS->H.sUnit;
           } /* if */
          rCPS->pxMullion=pMouse.x;
          if (rCPS->pxMullion==0 && rCPS->iFocusPane&1)
           {/* focus-pane is destroyed: reduce to 0 or 2 */
            rCPS->iFocusPane&=2;
           } /* if */
         } /* if */

        if (bVSet)
         {/* set new transom: check height of panes */
          if (pMouse.y<rCPS->V.sHead+
              rCPS->V.sUnit-rCPS->V.sMedian)
           {/* top pane too small: destroy transom, use bottom */
            rCPS->V.iBase[0]=rCPS->V.iBase[1];
            pMouse.y=0;
           }
```

► *Listing 2:*
(Fortsetzung)

```
        else if (pMouse.y>=wrc.bottom-syHScroll-rCPS->V.sUnit+
                 rCPS->V.sMedian)
         {/* bottom pane too small: destroy transom, use left */
          pMouse.y=0;
         }
        else if (rCPS->pyTransom==0)
         {/* new pane: determine index of base unit in pane */
          rCPS->V.iBase[1]=rCPS->V.iBase[0]+(pMouse.y-
               rCPS->V.sHead+rCPS->V.sMedian)/rCPS->V.sUnit;
         } /* if */
        rCPS->pyTransom=pMouse.y;
        if (rCPS->pyTransom==0 && rCPS->iFocusPane>=21)
         {/* focus-pane is destroyed: reduce to 0 or 1 */
          rCPS->iFocusPane-=2;
         } /* if */
       } /* if */
      SetPaneWindow(wrc.right,wrc.bottom);
     } /* if */
    bHSet=FALSE; bVSet=FALSE; bVChange=FALSE; bHChange=FALSE;
    ReleaseCapture(); bSplitting=FALSE; bKeySplit=FALSE;
   } /* if */
  if (bHSet && bHChange)
   {/* draw new mullion */
    pSplit.x=pMouse.x; /* set new point */
    PatBlt(hdc,pSplit.x,0,sxMullion,wrc.bottom,PATINVERT);
   } /* if */
  if (bVSet && bVChange)
   {/* draw transom selection rectangle */
    pSplit.y=pMouse.y; /* set new point */
    PatBlt(hdc,0,pSplit.y,wrc.right,syTransom,PATINVERT);
   } /* if */
  ReleaseDC(hwPane,hdc);
 } /* if */
 return 0L;
} /* case */
case WM_KEYDOWN:
 {RECT wrc;
  HWND hwPane=GetParent(hw);
  POINT pMouse,pMouseSet;
  BOOL bCtrl=GetKeyState(VK_CONTROL)>>15;
  /* abort if keyboard interface not active */
  if (!bKeySplit) return 0L;
  GetClientRect(hwPane,&wrc);
  /* ——— analyse pressed key ——— */
  pMouse=pSplit; /* get last location */
  switch (uP1)
   {case VK_LEFT:
      if (bCtrl) pMouse.x-=1;
      else
       {pMouse.x=
          max((pMouse.x-rCPS->H.sHead+rCPS->H.sMedian)/
              rCPS->H.sUnit-1,0)
          *rCPS->H.sUnit+rCPS->H.sHead-rCPS->H.sMedian;
       } /* if */
      break;
    case VK_RIGHT:
      if (bCtrl) pMouse.x+=1;
      else
       {pMouse.x=((pMouse.x-rCPS->H.sHead+rCPS->H.sMedian)/
                  rCPS->H.sUnit+1)
                 *rCPS->H.sUnit+rCPS->H.sHead-rCPS->H.sMedian;
       } /* if */
      break;
    case VK_UP:
      if (bCtrl) pMouse.y-=1;
      else
       {pMouse.y=
          max((pMouse.y-rCPS->V.sHead+rCPS->V.sMedian)/
              rCPS->V.sUnit-1,0)
          *rCPS->V.sUnit+rCPS->V.sHead-rCPS->V.sMedian;
       } /* if */
      break;
    case VK_DOWN:
      if (bCtrl) pMouse.y+=1;
      else
       {pMouse.y=((pMouse.y-rCPS->V.sHead+rCPS->V.sMedian)/
                  rCPS->V.sUnit+1)
                 *rCPS->V.sUnit+rCPS->V.sHead-rCPS->V.sMedian;
       } /* if */
      break;
    case VK_HOME:
      pMouse.x=rCPS->H.sHead; pMouse.y=rCPS->V.sHead; /* origin */
      break;
    case VK_END:
      pMouse.x=32767; pMouse.y=32767; /* end location */
      break;
    case VK_PRIOR:
      /* set left column or top line */
      if (bCtrl) pMouse.x=rCPS->H.sHead; else
        pMouse.y=rCPS->V.sHead;
    case VK_NEXT:
      /* set right column or bottom line */
      if (bCtrl) pMouse.x=32767; else pMouse.y=32767;
      break;
    case VK_ESCAPE:
      /* ignore new setting (set cross bars to old values */
      bSetSplit=TRUE;
      pMouse.x=rCPS->pxMullion; pMouse.y=rCPS->pyTransom;
      break;
    case VK_RETURN:
      /* use new setting */
      bSetSplit=TRUE; break;
   } /* switch */
  LimitSplitPos(&wrc,&pMouse);
  /* set new mouse position */
  pMouseSet.x=pMouse.x+sxMouseOffs;
  pMouseSet.y=pMouse.y+syMouseOffs;
  ClientToScreen(hwPane,&pMouseSet);
  /* set new position (and send WM_MOUSEMOVE command) */
  SetCursorPos(pMouseSet.x,pMouseSet.y);
 } /* case */
} /* switch */
return DefWindowProc(hw,iMsg,uP1,ulP2);
} /* fwSplit() */

/**********************************************************************
C h a n g e P a n e S i z e
=======================================================================

This function change the size of the panes as a result of calling the
"split" command in the application menu. The user interface is full
CUA-compatible. The size of the panes can be changed by keyboard or
mouse.

Parameters:
 none
Return:
 none

**********************************************************************/
```

Guide von [1], in dem ähnlicher Quellcode beschrieben ist.

Ein kleines Problem bei der Programmierung mit MS-Windows ist der Zugriff auf Dateien. Dies ist die einzige Ein-/Ausgabe-Schnittstelle eines Programms, die nicht durch Windows-API-Funktionen unterstützt wird. Lediglich die Funktionen `OpenFile`, mit der Dateien geöffnet, angelegt, gesucht oder gelöscht werden können, stellt hier eine gewisse Ausnahme dar. Die Windows-Dokumentation [1] empfiehlt, die Dateiausgabe durch Aufrufe der entsprechenden DOS-Interrupts zu realisieren. Weitere Hinweise werden keine gegeben. Neben der Möglichkeit, eigene Assembler-Module für den DOS-Zugriff mittels Interrupt 21h zu schreiben, können prinzipiell auch die Dateifunktionen des C-Compilers verwendet werden. Sie sind aber aus Kompatiblitätsgründen gegenüber UNIX, von dem diese Standardfunktionen stammen, verhältnismäßig ineffizient. Innerhalb von Windows existieren aber einige Low-Level-Dateizugriffsfunktionen, die beim Toolkit zu Windows 1.0x noch in der Datei WINEXP.H aufgeführt waren, beim SDK zu Windows 2 aber irgendwie verschwunden sind. Irgendwo wird auch erwähnt, diese Funktionen besser nicht zu verwenden, da sie in zukünftigen Versionen nicht mehr existieren könnten. Andererseits wird in einer Reihe von Microsoft-Windows-Applikationen gerade auf diese Applikationen zugegriffen, so daß es unwahrscheinlich ist, daß sie aus dem Windows-Kern entfernt werden. Die Datei WINEXP.H wurde übrigens auch beim SDK-Windows 2 mitgeliefert und befindet sich im Verzeichnis `\CARDFILE` auf der Diskette »Sample Source Code«.

Wenn man sich den Code der Datei-Funktionen mit CodeView ansieht, stellt man fest, daß sie sehr elementar programmiert sind und im wesentlichen nur die entsprechende INT-21H-DOS-Funktion aufrufen. Die verwendeten Funktionen wurden mit ihren Definitionsköpfen noch einmal am Anfang des FILE-Moduls aufgeführt, sie beginnen alle mit »_l«. In der Funktion `LoadFile` wird die Funktion `_llseek` dazu verwendet, die Größe der Datei zu bestimmen, indem mit dem dritten Parameter 2 das Dateiende positioniert und dessen Positionswert zurückgegeben wird.

Implementierung mit dem Medium-Speichermodell

Die Applikation DUMP ist bereits verhältnismäßig lang. Daher wurden die drei Module nicht zu einem Code-Segment zusammengefaßt, sondern innerhalb der Programmdatei DUMP.EXE als drei von der Speicherverwaltung unabhängig verwaltbare Segmente implementiert. Voraussetzung hierfür ist lediglich, daß beim Übersetzen die Optionen »-AM« und »-NT *ModulName*« ange-

geben werden und beim Linken die Bibliotheken für das Medium-Speichermodell angegeben werden. In der Linker-Definitionsdatei müssen dann für die einzelnen Module noch einmal aufgeführt werden und jedem Namen die Segment-Verwaltungsoptionen angegeben werden.

Die Anordnung der Funktionen in den drei Modulen DUMP, PANE und FILE wurde aus didaktischen Gründen vorgenommen. Für die Laufzeit der Applikation ist die Anordnung keinesfalls optimal, da zu stark zwischen den einzelnen Modulen hin- und hergesprungen werden muß, und daher die Speicherorganisation ungünstig werden kann. Bei kleinen Programmen wie DUMP ist dies sicherlich tolerierbar, aber bei großen Applikationen wie MS-EXCEL oder Aldus PageMaker ist die Organisation der Module eine Wissenschaft für sich und erfordert viel Erfahrung. Beim begrenzten Speicherplatz von Windows stellt diese Organisation aber die Grundvoraussetzung dafür dar, daß sich eine Applikation akzeptabel schnell bedienen läßt. Dies gilt insbesondere beim Bildaufbau oder der Zeicheneingabe. Der Bildaufbau kann noch so gut durchgeplant sein, die Geschwindigkeit sinkt hoffnungslos, wenn beispielsweise für jede auszugebende Zeile 20 oder mehr Segmente aufgerufen werden müssen und sich diese nicht alle gleichzeitig im zu kleinen Speicher befinden können. Als Faustregel gilt daher, einerseits die Modulgröße nicht zu groß werden zu lassen und andererseits nicht zu stark zwischen Modulen hin- und herzuspringen. Es ist oft günstiger, den Code einer Funktion in mehreren Modulen parallel zu halten, als stattdessen den Code in einem zentralen Modul abzulegen, welches dann seinerseits weitere Modulaufrufe nach sich zieht. Solche Überlegungen werden natürlich wirklich erst bei größeren Applikationen bedeutend. Bedenken Sie aber, daß beispielsweise Aldus PageMaker 178 Segmente und MS-Excel gar 187 Segmente besitzt. Ad-Hoc-Lösungen dürfen bei solchen Programmgrößen kaum zum effizienten Ablauf führen.

Ein weiteres Problem stellt das Debugging von Applikationen im Medium-Modell mit CodeView oder SymDeb dar. Beide Debugger können zwar Haltepunkte auf nicht geladenen Code setzen, aber sie können die benötigten Segmente nicht laden. Das ist ungünstig, wenn man sich etwa den Code nach dem Laden der Applikation erst einmal ansehen möchte, bevor man Haltepunkte setzt. Dies funktioniert erst dann, nachdem eine Funktion im entsprechenden Segment von der Applikation aufgerufen worden ist. Ein weiteres Problem stellt der »Trace into«-Befehl von *CodeView* bei verschiebbaren Segmenten dar. Manchmal vergißt der Befehl nach der Rückkehr von einer Funktion die Rücksprungadresse und fährt einfach fort, als hätte man »Step over« eingegeben. Und wieder ist die Applikation abgestürzt...

◄ *Listing 2: (Fortsetzung)*

```
VOID ChangePaneSize(VOID)

 {SetPaneFocus(-1,FALSE); /* switch off caret */
  SetFocus(rCPS->hwVSplit); /* change focus */
  bKeySplit=TRUE; /* activate keyboard input */
  /* send to split window for setting default-panes (middled) */
  SendMessage(rCPS->hwVSplit,SPM_SPLIT,0,0L);
 } /* ChangePaneSize() */


/**********************************************************************
I n i t P a n e M a n a g e r
======================================================================

This function initializes the complete pane manager.

Parameters:
 hiPrev  is the handle of the previous instance of NULL.

Return:
 TRUE if pane manager completely created otherwise FALSE.

**********************************************************************/

BOOL InitPaneManager(HANDLE hiPrev)

 {WNDCLASS wwc;

  if (!hiPrev)
   {/* —— create window class of split control area —— */
    wwc.lpszClassName="Split"; wwc.hInstance=hiMain;
    wwc.lpfnWndProc=fwSplit;
    wwc.hCursor=NULL;
    wwc.hbrBackground=COLOR_WINDOW+1;
    wwc.style=CS_DBLCLKS; wwc.hIcon=NULL; wwc.lpszMenuName=NULL;
    wwc.cbClsExtra=0; wwc.cbWndExtra=0;
    /* register window, return if error */
    if (!RegisterClass(&wwc)) return FALSE;
   } /* if */
  /* determine system-specific constants */
  sxMullion=(GetSystemMetrics(SM_CXHTHUMB)+2)/4;
  syTransom=(GetSystemMetrics(SM_CYVTHUMB)+2)/4;
  syHScroll=GetSystemMetrics(SM_CYHSCROLL);
  sxVScroll=GetSystemMetrics(SM_CXVSCROLL);
  hcrSplitH=LoadCursor(hiMain,MAKEINTRESOURCE(CRS_SPLITH));
  hcrSplitV=LoadCursor(hiMain,MAKEINTRESOURCE(CRS_SPLITV));
  hcrSplitX=LoadCursor(hiMain,MAKEINTRESOURCE(CRS_SPLITX));
  return TRUE;
 } /* InitPaneManager() */


/**********************************************************************
 C r e a t e P a n e W i n d o w s
======================================================================

This function creates the pane child windows for the pane
specification <rCPS>.

Parameters:
 none

Used Globals:
 rCPS

Return:
 TRUE if the pane windows are crated, otherwise FALSE.

**********************************************************************/

BOOL CreatePaneWindows(VOID)

 {int i;

  /* reset paning, invalidate pane sizes */
  rCPS->pxMullion=0; rCPS->pyTransom=0; rCPS->Pane[0].wrc.left!=-1;
  rCPS->iFocusPane=0;
  /* create window of horizontal split field */
  rCPS->hwHSplit=CreateWindow
   ("Split","",WS_CHILD|SPS_HORZ,0,0,0,0,
    rCPS->hwPane,IDHSPLIT,hiMain,NULL);
  if (!rCPS->hwHSplit) return FALSE;
  /* create window of vertical split field */
  rCPS->hwVSplit=CreateWindow
   ("Split","",WS_CHILD|SPS_VERT,0,0,0,0,
    rCPS->hwPane,IDVSPLIT,hiMain,NULL);
  if (!rCPS->hwVSplit) return FALSE;
  /* create scroll bar childs for pane window */
  for (i=0;i<4;i++)
   {rCPS->Scroll[i].hw=CreateWindow
     ("ScrollBar","",WS_CHILD|(i&1? SBS_HORZ:SBS_VERT),
      0,0,0,0,rCPS->hwPane,IDPSCROLL+i,hiMain,NULL);
    if (!rCPS->Scroll[i].hw) return FALSE;
   } /* for */
  return TRUE;
 } /* CreatePaneWindows() */


/**********************************************************************
C l o s e P a n e s
======================================================================

This function closes the panes of the pane specification <rCPS> and
destroys all child windows of the pane window. The pane manager is no
longer valid for this window and <rCPS> is set to NULL.

Parameters:
 none

Used Globals:
 rCPS

Return:
 none

**********************************************************************/

VOID ClosePanes(VOID)

 {int i;
  if (GetFocus()==rCPS->hwPane && rCPS->Pane[0].wrc.left!=-1)
   {/* turn off pane focus */
    SetPaneFocus(-1,FALSE);
   } /* if */

  DestroyWindow(rCPS->hwHSplit); DestroyWindow(rCPS->hwVSplit);
  for (i=0;i<4;i++)
   {/* destroy scroll bar windows */
    DestroyWindow(rCPS->Scroll[i].hw);
   } /* for */
  rCPS=NULL;
 } /* ClosePanes() */
```



MICROSOFT
WINDOWS
PRESENTATION
MANAGER
MS/DOS



MICROSOFT EXCEL
Das fortschrittliche Planungssystem mit Grafik und Datenbank.
MS/DOS OS/2

Software heute präsentiert eine Leistungsdimension, die Neues schwer vorstellbar macht. Ansätze für wirkliche Innovationen bietet das »Konzept Mensch« – die Integration menschlicher Verhaltensweisen in Software-Konzepte, um damit zukunftsweisende Standards zu schaffen. GUI, das »Graphical User Interface«, ist ein solcher Standard, mit dem Microsoft alle Möglichkeiten der neuen leistungsfähigen PC-Generation ausschöpft. Gleichzeitig sind die Microsoft GUI-Systeme integraler Bestandteil der Software-Strategien aller bedeutenden EDV-Anbieter.

GUI akzeptiert den Menschen als »Augen-Wesen«, das Signale und Symbole schneller als Buchstaben erfaßt und umsetzt. GUI ist das Konzept einer völlig neu entwickelten, grafischen Benutzerführung, die in der MS-DOS-Welt unter MICROSOFT WINDOWS und in der MS OS/2-Welt unter dem MICROSOFT PRESENTATION MANAGER den weltweiten Standard setzt. Also jeder professionellen Anwendung entspricht. Damit sind Programme nur im Inhalt verschieden, nicht in der Anwendung. Vorteil: Schulungs- und Einarbeitungskosten werden deutlich reduziert, das Lernen wird durch voll integrierte »On-Line-Lernprogramme« zusätzlich vereinfacht und das Zusammenwirken der Programme durch dynamischen Datenaustausch (DDE) optimiert.

MICROSOFT WINDOWS PRESENTATION MANAGER ist damit der GUI-Integrationsstandard für die neue Generation der PC-Anwendungssoftware. Ihr Nutzen sind produktiver PC-Einsatz und motiviertere PC-Anwender. Verbunden mit der Gewißheit, auch in Zukunft für die neuesten Entwicklungen in der PC-Software-Technologie offen zu sein.

Microsoft®

ZUKUNFT DER SOFTWARE

▶ *Listing 2: (Ende)*

```
/***********************************************************************
 P r o c e s s P a n e M s g
========================================================================

This function is called if the pane window has received a message. All
usable messages are analysed if processed if possible. The function
returns 1L if the message was processed completely and 0L if not.
This function may be called only if <rCPS> is valid for <hw>.

Parameters:
 see standard window function

Return:
 none

***********************************************************************/

LONG ProcessPaneMsg(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {int iPane;
  PANE *rPane;
  RECT wrc;
  short iSavedDC;

  switch (iMsg)
   {case WM_SIZE:
      /* resize interior of pane window */
      SetPaneWindow(P2LO,P2HI);
      return 0L;
    case WM_PAINT:
     {HDC hdc;
      PAINTSTRUCT wps;
      int vRegion;
      SetPaneFocus(-1,FALSE);
      hdc=BeginPaint(hw,&wps);
      CreateDrawingTools(hdc);
      for (iPane=0,rPane=rCPS->Pane;iPane<4;iPane++,rPane++)
       {/* determine if <iPane> is valid */
        if (!rPane->bValid) continue; /* invalid pane */
        /* save display context, reduce clipping rectangle to pane */
        iSavedDC=SaveDC(hdc);
        vRegion=IntersectClipRect
         (hdc,rPane->wrc.left,rPane->wrc.top,
          rPane->wrc.right,rPane->wrc.bottom
         );
        if (vRegion!=NULLREGION)
         {/* set drawing viewport to pane */
          SetViewportOrg
           (hdc,rCPS->Pane[iPane].wrc.left,rCPS->Pane[iPane].wrc.top);
          /* call application specific drawing function */
          DrawPane
           (hdc,iPane,rPane->wrc.right-rPane->wrc.left,
            rPane->wrc.bottom-rPane->wrc.top
           );
         } /* if */
        /* restore prev. device context (with full clipping range) */
        RestoreDC(hdc,iSavedDC);
       } /* for */
      EndPaint(hw,&wps); DestroyDrawingTools();
      SetPaneFocus(rCPS->iFocusPane,FALSE);
      return 1L;
     } /* case */
    case WM_SETFOCUS:
      /* window gets the focus: set caret to active pane */
      SetPaneFocus(rCPS->iFocusPane,FALSE);
      break;
    case WM_KILLFOCUS:
      /* window loses the focus: hide caret */
       if (rCPS->Pane[0].wrc.left!=-1) SetPaneFocus(-1,FALSE);
      break;
    case WM_VSCROLL:

    case WM_HSCROLL:
      /* execute scrolling of panes */
      ScrollPane(GetWindowWord(P2HI,GWW_ID)-IDPSCROLL,uP1,P2LO);
      return 1L; /* consumed */

    case WM_KEYDOWN:
      /* ——— analyse if pane control character ——— */
      if (uP1==VK_F6)
       {/* —— switch from pane to pane —— */
        static BYTE miNextPane[]={1,3,0,2};
        static BYTE miPrevPane[]={2,0,3,1};
        int iNewPane=rCPS->iFocusPane;
        BOOL bPrev=GetKeyState(VK_SHIFT)>>15;
        do
         {iNewPane=bPrev? miPrevPane[iNewPane]:miNextPane[iNewPane];
          iNewPane&=3; /* reduce to 0..3 */
          if (rCPS->Pane[iNewPane].bValid) break;
         }
        while (iNewPane!=rCPS->iFocusPane);
        if (iNewPane!=rCPS->iFocusPane)
         {/* set new focus pane */
          rCPS->iFocusPane=iNewPane;
          SetPaneFocus(rCPS->iFocusPane,TRUE);
         } /* if */
        return 1L; /* consumed */
       } /* if */
      return 0L; /* not consumed */

    case WM_LBUTTONDOWN:
      /* set focus to clicked pane */
      if (GetFocus()!=rCPS->hwPane)
       {/* set focus to application, no caret moving */
        SetFocus(rCPS->hwPane); return 1L; /* consumed */
       }
      else
       {/* determine new pane */
        int i;
        for (i=0;i<4;i++)
         {if (rCPS->Pane[i].bValid &&
              PtInRect(&rCPS->Pane[i].wrc,MAKEPOINT(ulP2)))
           {/* set new pane */
            if (i!=rCPS->iFocusPane)
             {/* set <i> as new focus */
              rCPS->iFocusPane=i; SetPaneFocus(rCPS->iFocusPane,TRUE);
             } /* if */
            break;
           } /* if */
         } /* for */
       } /* if */
      break; /* not completely consumed */
   } /* switch */
  return 0L; /* not consumed */
 } /* ProcessPaneMsg() */


/* ==========================================
   Windows application DUMP: end of module PANE
   ==========================================
*/
```

Um dies zu verhindern, ist es empfehlenswert, die einzelnen Segmente mit den Speicheroptionen `PRELOAD` und `FIXED` zu versehen, anstatt wie üblich mit `LOADONCALL`, `DISCARDABLE` und `MOVABLE`. Um sowohl eine Debugging-Version als auch eine »Produktion«-Version der Applikation zu erhalten, wurden getrennte MAKE- und Linker-Definitionsdateien erzeugt. Die MAKE-Dateien tragen die Namen DUMP.MD (»Make Debug«) und DUMP.MP (»Make Product«) und befinden sich in *Listing 5* und *6*. Im ersteren Fall wird auch mit der Option »Zi« Code für den CodeView erzeugt. In DUMP.MP wird dagegen mit »Ox« der Optimierer eingeschaltet, der wegen möglicher Codeumlagerung beim Debugging zuviel Verwirrung stiften könnte. Die Linker-Einträge in den beiden Dateien rufen entsprechend die Definitionsdateien DUMP.DFD (*Listing 10*) und DUMP.DFP (*Listing 11*) auf.

Es wäre schön, wenn Microsoft den CodeView so ändern würde, daß er besser mit verschiebbaren Segmenten zurechtkommt und auch selbständig Segmente nachladen kann. Bis dahin ist man auf die Entwicklung mit zwei Varianten angewiesen.

Vorteile des Windows-Zeichenprinzips

Das Schöne an der Windows-Fensterverwaltung ist die Art, wie Fensterinhalte neu gezeichnet werden: Eine oder mehrere Instanzen legen fest, welcher Teil des Fensterinhalts sich geändert hat oder nicht mehr verdeckt ist, und eine einzige Instanz, die die WM_PAINT-Nachricht in der Fensterfunktion abfängt, zeichnet diese Teile neu. Als ich mich vor längerer Zeit in Windows eingearbeitet habe, war für mich dieses Prinzip zwar nach einiger Zeit geläufig, unklar blieb aber, was denn das ganze eigentlich soll. Schließlich könnte man ja gleich die geänderten Teile zeichnen, anstatt erst einmal komplexe Datenstrukturen aufzubauen, die in einem weiteren Schritt wieder interpretiert werden müssen. Doch beim Entwickeln des Pane-Managers wurde mir klar, welche Genialität hinter diesem Konzept steht. Die Behandlung der WM_PAINT-Nachricht stellt bei vielen Applikationen (man denke an Textverarbeitung, CAD oder Desktop-Publishing) einer der aufwendigsten und applikationsspezifischsten Teile überhaupt dar. Doch braucht man sich bei der Implementierung dieses Teils nicht darum zu kümmern, wie gerollt oder geblättert wird oder ob es Unterfenster gibt. Die Applikation zeichnet vielmehr stur die geänderten Bereiche. Die Zeichenausschnitt-Organisation stellt sicher, daß auch nur die ungültig gemachten Bereiche gezeichnet werden und Teile außerhalb davon nicht verändert werden. Völlig unabhängig davon lassen sich etwa eine Scrolling-Verwaltung oder eben der Pane-Manager appli-

kationsunabhängig realisieren. Dadurch wird die applikationsübergreifende Wiederverwendbarkeit von Software erheblich vereinfacht. Laut Bill Gates, dem Chairman von Microsoft, ist dies letztendlich die einzige (!) Technik, die die Software-Entwicklung tatsächlich beschleunigt. (»There's only one trick in software and that is using a piece of software that's already been written« [4]).

Mögliche Verbesserungen des Managers

Wie am Anfang des Artikels angedeutet wurde, kann die Anzahl der Unterfenster die Zahl 4 übersteigen. Für viele Fälle ist dies sinnvoll, insbesondere dann, wenn die Bildschirme größer werden und mehr Information gleichzeitig dargestellt werden kann. Die Erweiterung von zwei auf mehr Unterteilungen in der Vertikalen und Horizontalen ist nicht weiter schwierig, es führt aber zu einem deutlichen höheren Programmieraufwand als die vorgestellte Lösung.

Manchmal ist es nachteilig, daß sich eine Rolleiste auf alle in der gleichen Spalte oder Zeile plazierten Unterfenster bezieht. Dies kann dadurch umgangen werden, daß man einzelne Unterfenster mit Hilfe eines Befehls fixieren kann. Auch dies wurde bei MS-Excel mit dem Menü-Befehl »Fenster fixieren« realisiert, mit dem das obere und das linke Unterfenster »eingefroren« werden, so daß sich die Manipulationen der Rolleiste nur noch auf das rechte und das untere Unterfenster bezieht. Bei geübter Verwendung dieses Befehls kann man die Flexibilität der Unterfenster weiter erhöhen.

Das Gegenteil von fixierten Unterfenstern ist das Verbinden von Rolleisten von Unterfenstern. Hat man etwa eine Tabelle horizontal in zwei Hälften unterteilt und hat das untere Unterfenster an eine andere Position bewegt, möchte man vielleicht die Tabellenteile in den beiden Unterfenstern Zeile für Zeile vergleichen und dabei mit einer Anweisung an die Rolleiste in beiden Unterfenstern gleichzeitig rollen oder blättern. Dies kann mit einem Befehl »Fenster verbinden« oder ähnlich realisiert werden. Dieser nützliche Befehl fehlt bei MS-Excel leider noch.

Die im vorgegangenen Kapitel gemachten Bemerkungen sind wichtig für weitere Verbesserungen des Pane-Managers. Im Augenblick werden die bis zu vier Unterfenster unabhängig nacheinander gezeichnet. Man kann dies beschleunigen, indem ein Algorithmus feststellt, welche Dokumentteile in den einzelnen Unter fenstern mehrfach vorkommen und diese aus einem Unterfenster in die anderen kopieren. Dadurch wird unter Umständen der Aufwand für das eigentliche Neuzeichnen ähnlich wie beim Scrolling weiter reduziert. Ähnliche Algorithmen finden Verwendung, wenn es darum geht, Veränderungen in den Dokumenten, etwa bei der Editierung in einem Unterfenster, an alle Unterfenster weiterzuleiten. Auch hierzu werden sinnvollerweise Kopieraktionen Verwendung finden.

Auf den Einsatz solcher Algorithmen wurde beim vorliegenden Pane-Manager verzichtet, dies würde den Rahmen dieses Artikels sprengen. Wichtig ist aber, auch hier festzuhalten, daß solche Optimierungen in keinster Weise die applikationsspezifische Schnittstelle des Pane-Managers berühren, sondern davon verborgen im Kern des Managers eingebaut werden.

Ich hoffe, dem Leser mit diesem Artikel einige Anregungen für eigene Windows-Applikationen gegeben zu haben. Sie sollten jetzt wissen: Unterfenster wurden von Microsoft und IBM zum Windows-Standardstil erhoben und, wenn sie eine gewisse Verbreitung gefunden haben, wird sie kein Anwender bei der täglichen Arbeit mehr missen wollen. Bildschirmteiler und Fensterkreuz werden dann so selbstverständlich verwendet werden wie heute die Rolleisten. Sorgen Sie deshalb dafür, daß Ihre Kunden *nicht* über Ihr Produkt sagen müssen: »Ein Programm, das immer noch keine Unterfenster hat!«

Marcellus Buchheit

[1] Microsoft Windows System Development-Kit, Version 2.

[2] Welch, K.P.: Die Mehrfachdokumentenschnittstelle (MDI), Microsoft System Journal, Juli/August 1989, Seite 5 bis 33.

[3] Buchheit, Marcellus: Hilfe-Verwaltung für Windows, Microsoft System Journal, September/Oktober 1989, Seite 30 bis 57.

[4] Nelson, Robin: Software's Midlife Crisis – Interview with Bill Gates. Electronics, June 1989, Page 68-72.

[5] Petzold, Charles: Multi-Thread-Anwendungen unter MS-OS/2 1.1, Microsoft System Journal, Januar/Februar 1989, Seite 4 bis 17.

[6] Wettstein, H.: Architektur von Betriebssystemen, 3. Auflage 1987, Hanser-Verlag München, Wien.

[7] Adler, Marc: Eine virtuelle Speicherverwaltung in C, Microsoft System Journal, September/Oktober 1989, Seite 104 bis 115.

[8] Yao, Paul: Leistungssteigerung durch EMS, Microsoft System Journal, Januar/Februar 1989, Seite 74 bis 81.

Listing 3:
Das Modul der Datei-Verwaltung, FILE.C.

```
/*********************************************************************

 —— D U M P ———————— MS-Windows Application ————————————
                        Module FILE.C

======================================================================

This Module contains the file management and utility functions of the
DUMP application.

----------------------------------------------------------------------

Copyright 1989 by
         Marcellus Buchheit, Buchheit software research
         Zaehringerstrasse 47, D-7500 Karlsruhe 1
         Phone (0) 721/37 67 76    (West Germany)

Release 1.00 of 89-Sep-22 —— All rights reserved.

*********************************************************************/

/* read common header files */
#include "DUMP.H"

/* used low-level Windows disk-I/O functions */
int FAR PASCAL _lopen(LPSTR lpPathName,int iReadWrite);
int FAR PASCAL _lclose(int hFile);
LONG FAR PASCAL _llseek(int hFile,LONG lOffset,int iOrigin);
WORD FAR PASCAL _lread(int hFile,VOID FAR* lpBuffer,WORD wBytes);


/*********************************************************************

 ——————————————— File-Management-Variables ———————————————

*********************************************************************/

BYTE zCurrentFile[12+1]; /* name of current file (without path) */
BYTE zNewFile[12+1]; /* name of new file (specified by open dialog) */
int hfNewFile;       /* handle to new file, <=0 if file not opened */

/* —— file I/O dialog variables —— */
BYTE zCurrentPath[128];          /* current path specification */
BYTE zDefaultFileSpec[13]="*.*"; /* default file specification */
BYTE zDefaultExtend[5]=".*";   /* default extend string (all files) */


/*********************************************************************
 E r r o r N o M e m
======================================================================

The error message "Not enough memory" is displayed in a message box.

Parameters:
 hw  is the window handle of the next active window and NULL if no
     window exists.

Used variables:
 rzAppTitle  is the local handle to the application title name.
 rzNoMemory  is the local handle to the no-memory error string.

Return:
 None

*********************************************************************/

VOID ErrorNoMem(HWND hw)

 {MessageBox(hw,rzNoMemory,rzAppTitle,MB_ICONHAND|MB_OK);
 } /* ErrorNoMem() */


/*********************************************************************
 P r i n t M e s s a g e
======================================================================

The string with resource index <iString> is displayed in a message box
with an asterisk. The string can contain placeholder for a printf()-
call. Each of this placeholder is replaced by the arguments which
follows after <iString> in the variable length argument of
PrintMessage().

Parameters:
 iCtrl  is the index of the control string.
Further parameters can follow and specify the replacing

Used variables:
 rzAppTitle  is the pointer to the application title name.

Return:
 None

*********************************************************************/

VOID PrintMessage(WORD iString,...)

 {BYTE zMsg[256],zString[128];
  va_list rVarArg;

  /* load string from resource */
  LoadString(hiMain,iString,zString,sizeof(zString));
  va_start(rVarArg,iString); /* set rVarArg to first option param. */
  vsprintf(zMsg,zString,rVarArg); /* create message string */
  va_end(rVarArg);
  /* display message */
  MessageBox(NULL,zMsg,rzAppTitle,MB_ICONASTERISK|MB_OK);
 } /* PrintMessage() */


/*********************************************************************
 R e a d D i a l o g
======================================================================

This function creates a modeless dialog box, displayes it at screen
and returns if the user closes the box.
The function returns TRUE if the function DialogBox() returns IDOK and
FALSE otherwise. If the dialog box is not created because the memory
is too small, a message box with an error is created and FALSE is
returned.

Parameters:
 uBox  is the number of the dialog box (DBX_...).
 rfd   is the address of the dialog box function.

Return:
 TRUE if the dialog box was created and the OK-button was pressed to
 close it or FALSE if the CANCEL-button was pressed or no box was
 created.

*********************************************************************/

BOOL ReadDialog(WORD uBox,FARPROC rfd)

 {FARPROC rfdInst;
  int v;

  rfdInst=MakeProcInstance(rfd,hiMain);
  if (rfdInst!=NULL)
   {v=DialogBox(hiMain,MAKEINTRESOURCE(uBox),hwMain,rfdInst);
    FreeProcInstance(rfdInst);
    if (v!=-1)
     {/* box was created: return TRUE if OK-button pressed */
      return v==IDOK;
     } /* if */
   } /* if */
  /* procedure-instance or box not created: print error message */
  ErrorNoMem(hwMain); return FALSE;
 } /* ReadDialog() */


/*********************************************************************
 L o a d F i l e
======================================================================

This function analyses the length of the opened file <hf> and
allocates the memory for the virtual reading of the file data.

Parameters:
 rSpec  is the address of the file specification of the current file.

Return:
 TRUE is returned if the file was read correctly and the memory was
 allocated or FALSE otherwise.

*********************************************************************/

BOOL LoadFile(FILESPEC *rSpec)

 {LONG vSize; /* size of file */
  HANDLE hmg;

  vSize=_llseek(rSpec->hf,0L,2); /* read last position = length */
  if (vSize>0x4000000L) /* 64 MBytes */
   {/* file too long: error message, return FALSE */
    PrintMessage(ST_TOO_LONG,zCurrentFile);
    return FALSE;
   } /* if */
  rSpec->vSize=vSize; /* set into specification */
  /* calc. size of data block header (one handle per 4 KB file size */
  rSpec->seHead=(WORD)((vSize+4095L)/4096);
  /* allocate memory block (init. with 0 => no data area loaded) */
  hmg=GlobalAlloc
   (GMEM_MOVEABLE|GMEM_ZEROINIT,(LONG)(sizeof(HANDLE)*rSpec->seHead));
  if (!hmg)
   {/* memory too small: error message, return FALSE */
    ErrorNoMem(hwMain); return FALSE;
   } /* if */
  rSpec->hmgHead=hmg; /* set handle */
  return TRUE; /* no error */
 } /* LoadFile() */


/*********************************************************************
 R e a d D a t a B l o c k
======================================================================

This function reads the 4KB data block of address <uaFile> from the
file with specification <rSpec> and returns the handle of this data
block. This block must be locked immediately to avoid that the system
discards the block. The function analyses if the 4KB-block is already
loaded. If so, no new reload operation is started. Otherwise the
contents of the block is read from disk.

Parameters:
 rSpec   is the address of the file specification of the current file.
 uaFile  is the address value of the needed data.

Return:
 The valid handle of the global block is returned or NULL if the data
 are not read. In the last case a message box was displayed which has
 informated the user.

*********************************************************************/

HANDLE ReadDataBlock(FILESPEC *rSpec,DWORD uaFile)

 {WORD iBlock;
  HANDLE FAR *rhBlock;
  HANDLE FAR *rhHead;
  HANDLE FAR *rh;
  HANDLE hmgBlock;
  BYTE FAR *rdBlock;
  int v;

  /* calculate block number from address */
  iBlock=(WORD)(uaFile/4096);
  rhHead=(HANDLE FAR*)GlobalLock(rSpec->hmgHead);
  rh=rhHead+iBlock;
  hmgBlock=*rh;
  if (hmgBlock==NULL || (GlobalFlags(hmgBlock)&GMEM_DISCARDED))
   {/* ——— (re-)load data block from disk ——— */
    if (hmgBlock) GlobalFree(hmgBlock); /* free discarded block */
    hmgBlock=GlobalAlloc
     (GMEM_MOVEABLE|GMEM_DISCARDABLE|GMEM_NOTIFY,4096L);
    if (!hmgBlock)
     {/* memory too small: error message, return FALSE */
      ErrorNoMem(hwMain); goto RETURN;
     } /* if */
    /* set read location to start of 4-KB-block */
    _llseek(rSpec->hf,uaFile & 0xFFFFF000L,0);
    /* read data from file into data block (4 KB or less) */
    rdBlock=(BYTE FAR*)GlobalLock(hmgBlock);
    v=_lread(rSpec->hf,rdBlock,4096);
    GlobalUnlock(hmgBlock);
    if (v==-1)
     {/* error during read */
      PrintMessage(ST_BAD_READ,zCurrentFile); goto RETURN;
     } /* if */
    *rh=hmgBlock; /* set handle of allocated block */
   } /* if */
  RETURN:
  GlobalUnlock(rSpec->hmgHead);
  return hmgBlock;
 } /* ReadDataBlock() */


/*********************************************************************
 C l o s e F i l e
======================================================================

This function closes the opened file with specification <rSpec>. All
allocated memory is freed.
```

Listing 3:
(Fortsetzung)

```
Parameters:
 rSpec  is the address of the file specification.

Return:
 none

*********************************************************************/



VOID CloseFile(FILESPEC *rSpec)

 {int i;

   lclose(rSpec->hf); /* close file */
  if (rSpec->hmgHead)
   {/* ---- free all allocated data ---- */
    HANDLE FAR *rhmg;
    HANDLE FAR *rhHead;
    rhHead=(HANDLE FAR*)GlobalLock(rSpec->hmgHead);
    rhmg=rhHead;
    for (i=0;i<rSpec->seHead;i++,rhmg++)
     {if (*rhmg!=NULL)
       {/* free allocated data */
        GlobalFree(*rhmg);
       } /* if */
     } /* for */
    /* free data of header */
    GlobalUnlock(rSpec->hmgHead); GlobalFree(rSpec->hmgHead);
    rSpec->hmgHead=NULL; /* no header */
   } /* if */
 } /* CloseFile() */


/*********************************************************************

 ------------------ Open-File dialog box functions ------------------

*********************************************************************/

/*********************************************************************
 C h a n g e D e f a u l t E x t e n d
=====================================================================

The extension of the file specification <rzFile> is used as new
default file extend if it exists and if it contains no wildcards.
This default extention is stored in <zDefaultExtend>.

Parameters:
 rzFile  is the address of string which contains a valid file
         specification.

Used Variables:
 zDefaultExtend

Return:
 none

*********************************************************************/

VOID ChangeDefaultExtend(PSTR rzFile)

 {PSTR rz;

  rz=strchr(rzFile,'.');
  if (rz!=NULL && !strpbrk(rz,"*?")) strcpy(zDefaultExtend,rz);
 } /* ChangeDefaultExtend() */


/*********************************************************************
 A d d D e f a u l t E x t e n d
=====================================================================

The extension of the file specification <zFile> is set by
<zDefaultExtend> if <rzFile> contains no extend (no '.' in the
specification).

Parameters:
 rzFile  is the address of string which contains a valid file
         specification.

Used Variables:
 zDefaultExtend

Return:
 none

*********************************************************************/

VOID AddDefaultExtend(PSTR rzFile)

 {PSTR rz;

  /* determine if <rzFile> contains extend */
  rz=rzFile+strlen(rzFile)-1;
  while (*rz!=':' && *rz!='\\' && rz>rzFile && *rz!='.')
   {rz=(PSTR)AnsiPrev(rzFile,rz);
   } /* while */
  if (*rz!='.')
   {/* add extension */
    strcat(rzFile,zDefaultExtend);
   } /* if */
 } /* AddDefaultExtend() */


/*********************************************************************
 C h e c k F i l e E d i t
=====================================================================

It is checked if the edit field of the dialog box with windows <hw>
contains characters. If so the IDOK item of the dialog box is enabled.
If the field is empty it is disabled. A disabling/enabling process is
only executed if the state changes. This is controlled by the flag at
address <rbEnabled>.

Parameters:
 hw ....... is the window handle of the dialog box.
 rbEnabled  is the address of the enable flag.

Return:
 none

*********************************************************************/

VOID CheckFileEdit(HWND hw,BOOL *rbEnabled)

 {if ((SendMessage(GetDlgItem(hw,IDFILENAME),
      WM_GETTEXTLENGTH,0,0L)!=0)!=*rbEnabled)
   {/* change enable state */
    *rbEnabled=!*rbEnabled;
    EnableWindow(GetDlgItem(hw,IDOK),*rbEnabled);
   } /* if */
 } /* CheckFileEdit() */


/*********************************************************************
 S e p a r a t e F i l e N a m e
=====================================================================

The file name location of file specification <rzFile> is determined
and this address is returned.

Parameters:
 rzFile  is the address of string which contains a valid file
         specification.

Return:
 The address of the file name location in the file specification is
 returned.

*********************************************************************/

PSTR SeparateFileName(PSTR rzFile)

 {PSTR rz;

  /* search from end of string */
  rz=rzFile+strlen(rzFile)-1;
  while (*rz!=':' && *rz!='\\' && rz>rzFile)
        rz=(PSTR)AnsiPrev(rzFile,rz);
  if (rz!=rzFile) rz=(PSTR)AnsiNext(rz);
  return rz; /* address of first character */
 } /* SeparateFileName() */


/*********************************************************************
 f d O p e n F i l e
=====================================================================

### Dialog Box function ###

This function processes any messages received by the "OpenFile" dialog
box.

Parameters:
standard message data (see fwMain)

Further input:
The desired directory is set and after return the last displayed
directory is set (also if CANCEL).

Return:
standard dialog box function value
DialogBox() returns the code of the pressed terminate button
(IDOK or IDCANCEL).

*********************************************************************/

BOOL FAR PASCAL fdOpenFile(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)
 {static BOOL bEnabled;
  char z[128+5];
  char zSlct[24+1];
  PSTR rz;
  WORD u;

  switch (iMsg)
   {case WM_INITDIALOG:
      /* set contents of file list box (at current directory) */
      DlgDirList(hw,zDefaultFileSpec,IDFILELIST,IDPATH,0);
      DlgDirList(hw,zDefaultFileSpec,IDDIRLIST,NULL,0xC010);
      SetDlgItemText(hw,IDFILENAME,zDefaultFileSpec);
      /* select file entry complete */
      SendDlgItemMessage(hw,IDFILENAME,EM_SETSEL,0,MAKELONG(0,32767));
      /* enable/disable "open" item depending filename exists */
      bEnabled=TRUE; CheckFileEdit(hw,&bEnabled);
      SetFocus(GetDlgItem(hw,IDFILENAME));
      return FALSE; /* focus is set */
    case WM_COMMAND:
      switch (uP1)
       {/* analyse box item */
        case IDFILENAME:
          if (HIWORD(ulP2)==EN_CHANGE)
           {/* activate/deactive "open" item */
            CheckFileEdit(hw,&bEnabled);
           } /* if */
          return TRUE;
        case IDFILELIST:
        case IDDIRLIST:
          switch (HIWORD(ulP2))
           {case LBN_SELCHANGE:
              /* get selected entry (file, directory, drive) */
              DlgDirSelect(hw,zSlct,uP1);
              if (uP1==IDDIRLIST)
               {/* specify new directory/drive */
                GetDlgItemText(hw,IDFILENAME,z,sizeof(z));
                rz=SeparateFileName(z);
                if (*rz==0||!strpbrk(rz,"*?"))
                 {/* no file select entry: set default */
                  rz=zDefaultFileSpec;
                 } /* if */
                strcat(zSlct,rz); /* add selection mask */
               } /* if */
              SetDlgItemText(hw,IDFILENAME,zSlct);
              /* select file entry complete */
              SendDlgItemMessage
               (hw,IDFILENAME,EM_SETSEL,0,MAKELONG(0,32767));
              /* unselect other table */
              SendDlgItemMessage
               (hw,uP1==IDFILELIST? IDDIRLIST:IDFILELIST,
                LB_SETCURSEL,-1,0L);
              break;
            case LBN_DBLCLK:
              /* first click has transf. selection, now like IDOK */
              uP1=IDOK; goto OPEN_FILE;
           } /* switch */
          return TRUE;
        case IDOK:
          /* return if "open" field invalid */
          if (!bEnabled) return TRUE;
        OPEN_FILE:
          /* get contents of file edit line */
          GetDlgItemText(hw,IDFILENAME,z,128); AnsiUpper(z);
          if (DlgDirList(hw,z,IDDIRLIST,IDPATH,0xC010))
           {/* specification is directory: set new file list */
            AddDefaultExtend(z);
            DlgDirList(hw,z,IDFILELIST,NULL,0);
            SetDlgItemText(hw,IDFILENAME,z);
            /* set new default specification */
            strcpy(zDefaultFileSpec,z);
            return TRUE;
           } /* if */
```

► *Listing 3: (Ende)*

```
        /* —— Single specified file: open it —— */
        AddDefaultExtend(z); /* add extend if not specified */
        strcpy(zNewFile,SeparateFileName(z)); /* det. file spec. */
        hfNewFile=_lopen(zNewFile,OF_READ);
        if (hfNewFile<0)
         {/* not opened: print error, continue */
          if (strpbrk(zNewFile,"*?"))
           {/* bad directory: generate beep */
            MessageBeep(0);
           }
          else
           {/* file not found: error message */
            PrintMessage(ST_NO_FILE,zNewFile);
           } /* if */
          /* set focus to file line, select full line */
          SetFocus(GetDlgItem(hw,IDFILENAME));
          SendDlgItemMessage
           (hw,IDFILENAME,EM_SETSEL,0,MAKELONG(0,32767));
          /* don't abort box */
          break;
         } /* if */
        /* terminate dialog */
      case IDCANCEL:
        EndDialog(hw,uP1);
        break;
     } /* switch */
    return TRUE;
  } /* switch */
 return FALSE;
} /* fdOpenFile() */


/* ===========================================
   Windows application DUMP: end of module FILE
   ===========================================
*/
```

► *Listing 4: Die C-Kopfdatei DUMP.H mit den exportierten Pane-Manager-Declarationen.*

```
/*******************************************************************

 —— D U M P ———————— MS-Windows Application ——————————
                           ### Header file

===================================================================

This MS-Windows application displayes the data of a MS-DOS file as a
hexadecimal table. The application uses 4 panes within its application
window.

———————————————————————————————————————————————————————————————————

Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

Release 1.00 of 89-Sep-13 —— All rights reserved.

*******************************************************************/

/* define/undefine non-debugging option name */
#define NDEBUG

#define NOMINMAX
#include <WINDOWS.H>
#include "DEFS.H"

/* further C standard headers */
#include <STDLIB.H>
#include <STRING.H>
#include <STDIO.H>
#include <STDARG.H>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)
#define LADDR(r) (LONG)(LPSTR)(r)


/* ================================================================
   FILESPEC                                File specification for DUMP
   ————————————————————————————————————————————————————————————————
   vSize ... contains the length of the files in bytes.
   seHead    size of header in entries
   hmgHead   is the handle of the global memory block which contains
             the handles of the 4-KB-data-blocks.
   seHead    size of header in entries
   hf ...... is the handle of the opened file.
   ================================================================
*/

typedef struct
 {LONG vSize;
  HANDLE hmgHead;
  WORD seHead;
  HANDLE hf;
 } FILESPEC;

/* ===========================
   application specific variables
   ===========================
*/
extern BYTE zAppName[];  /* application module name */
extern BYTE *rzAppTitle; /* pointer to application title name */
extern BYTE *rzNoMemory; /* pointer to error string "no memory" */
extern HANDLE hiMain;  /* handle to application instance */
extern HWND hwMain;    /* handle to main window */

/* global file module variables */
extern BYTE zCurrentFile[12+1]; /* name of file without path */
extern BYTE zNewFile[12+1]; /* name of new file from dialog box */
extern int hfNewFile; /* handle to new file, <=0 if file not opened */
extern FILESPEC FileSpec; /* file specification of the dump file */
extern HCURSOR hcrWait;   /* hourglass */

/* ====================================
   panes manager control types & variables
   ====================================
*/

/* ================================================================
   PANE                                 Single pane control specification
   ————————————————————————————————————————————————————————————————
   wrc ... determines the rectangle of the pane within the client area
           of the pane window. This rectangle includes the vertical
           headers of pane 0 and 2 and the horizontal headers of pane
           1 and 3.
   iVBase  is the index of the first vertical unit (line) which is
           displayed in the pane.
   iHBase  is the index of the first horizontal unit (character,
           column etc) which is displayed in the pane.
   iVPos   is the current vertical position in vertical units in the
           pane.
   iHPos   is the current horizontal position in horizontal units in
           the pane.
   bValid  is TRUE if pane is valid (displayed),
           FALSE if not (non-existent).
   The index values can be in range -1073741824..+1073741823
   (-2**30..2**30-1).
   ================================================================
*/

typedef struct
 {RECT wrc;
  LONG iVBase;
  LONG iHBase;
  LONG iVPos;
  LONG iHPos;
  BOOL bValid;
 } PANE;


/* ================================================================
   DATASPEC            Data specification for horizontal and vertical
   ————————————————————————————————————————————————————————————————
   iMin   is the long index of the first unit of the data object.
   iMax   is the long maximum value of the scroll range (including).
   iPos   is the index of the selected unit. This is the caret
          location.
   iBase  is an array for every pane in the specified direction.
          The first
          value is the index of the first unit in the left or top
          pane, the second value is the index of the first unit in
          the right or bottom pane.
   sUnit  is the size in device units of a base element in the window
          (for vertical the line height, for horizontal a width spec)
   sHead  is the size in device units of the head (for vertical is
          this a top line in pane 0 and 1, for horizontal is this a
          left column in pane 0 and 2).
   uExt   is the multiply factor for the conversation from scroll bar
          values to scroll values. The value can be in range 1..32768.

   Typical horizontal units are characters, cells or columns. Typical
   vertical units are lines. The value of <iMin>, <iMax> and <iPos>
   can be in range -1073741824..+1073741823 (-2**30..2**30-1).
   ================================================================
*/

typedef struct
 {LONG iMin;
  LONG iMax;
  LONG iPos;
  LONG iBase[2];
  WORD sUnit;
  WORD sMedian;
  WORD sHead;
  WORD uExt;
 } DATASPEC;


/* scroll bar specification (privat type) */
typedef struct
 {BOOL bVisible; /* scroll bar is visible */
  HWND hw;        /* handle of scroll bar window */
  int iMin;   /* minimum value of scroll bar */
  int iMax;   /* maximum value of scroll bar */
  int iPos;   /* current scrollbar value */
  int nUnits; /* units (lines, characters etc) per pane */
 } SCROLL;


/* ================================================================
   PANESPEC            Data specification for horizontal and vertical
   ————————————————————————————————————————————————————————————————
   hwPane ........ is the window which contains the panes.
   H ............. is the horiz. data specification of type DATASPEC.
   V ............. is the vert. data specification of type DATASPEC.
   wrcTrackRedraw  is the invalidation rectangle for redrawing of the
                   displayed panes.
   iFocusPane .... is the index of the pane which has the input focus
                   if the pane window has the input focus.
   pxMullion ..... is the relative horizontal pane location within the
                   client area of the pane window.
   pyTransom ..... is the relative vertical pane location within the
                   client area of the pane window.
   ================================================================
*/
typedef struct
 {HWND hwPane;
  DATASPEC H;
  DATASPEC V;
  RECT wrcTrackRedraw;
  int iFocusPane; /* pane which has the input focus */
  int pxMullion; /* relative horizontal pane location */
  int pyTransom; /* relative vertical pane location */
  PANE Pane[4];
  /* private used data structures */
  SCROLL Scroll[4];
  HWND hwHSplit;
  HWND hwVSplit;
 } PANESPEC;

extern PANESPEC *rCPS; /* pointer to current pane specification */
/* public functions in DUMP.C */
VOID DrawTrackPos(int iPane,LONG iTrackBase);
VOID CreateDrawingTools(HDC hdc);
VOID SetPaneFocus(int iPane,BOOL bChange);
VOID DrawPane(HDC hdc,int iPane,WORD sxPane,WORD syPane);
VOID DestroyDrawingTools(VOID);

/* public functions in FILE.C */
VOID ErrorNoMem(HWND hw);
VOID PrintMessage(WORD iString,...);
BOOL ReadDialog(WORD uBox,FARPROC rfd);
BOOL LoadFile(FILESPEC *rSpec);
HANDLE ReadDataBlock(FILESPEC *rSpec,DWORD uaFile);
VOID CloseFile(FILESPEC *rSpec);
BOOL FAR PASCAL fdOpenFile(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2);

/* public functions in PANE.C */
BOOL InitPaneManager(HANDLE hiPrev);
BOOL CreatePaneWindows(VOID);
VOID SetScrollValues(BOOL bRedraw);
VOID SetPaneWindow(int sxWindow,int syWindow);
VOID ClosePanes(VOID);
VOID ScrollPane(int iScroll,WORD uCmd,int iNewPos);
VOID ChangePaneSize(VOID);
LONG ProcessPaneMsg(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2);
```

Die Welt steht Ihnen offen!

Die ganze Welt der PC-Software steht Ihnen offen. Sie ist leistungsfähig und vielseitig. Aber zwischen der einen Software und der anderen liegen oft Welten: Die Vielfalt in der Bedieneroberfläche verursacht unnötigen Schulungsaufwand und bremst den Anwender, statt seine Arbeit zu beschleunigen. Er gerät mit ständig wechselnden Problemen in Konflikt. ComfoBridge ist die Möglichkeit, dem Anwender eine Bedieneroberfläche für alle gängigen Software-Produkte zu schaffen. ComfoBridge läuft über die Industriestandard-Oberfläche MS-Windows, die leicht zu bedienen ist und in Zukunft Bestand haben wird. Mit ComfoBridge sind Sie in der Lage, Bedienungsabläufe zu automatisieren, verschiedene Programme, auch gleichzeitig, zu steuern und eigene Menüs zu erstellen. ComfoBridge ist das Werkzeug, mit dem Sie all diese Aufgaben mit einem Höchstmaß an Programmierkomfort erledigen. Weltoffen!



ComfoBridge ist ein Baustein zum SPI-Windows-Office.
Weitere Bausteine sind
ComfoTalk, ComfoDesk
Clip & Connect
ComfoTex 2
ACCESS SQL

Bitte schicken Sie weitere Informationen über:
☐ Das SPI-Windows-Office ☐ ComfoBridge
☐ Die Open Access II Familie ☐ Das SQL System

Firma

Name/Funktion

Straße

PLZ/Ort

Telefon

Sys 10/89





SOFTWARE PRODUCTS INTERNATIONAL

SPI Software Products International (Deutschland) GmbH
Stefan-George-Ring 22+24, 8000 München 81
Tel.: 089/930090-0, Fax: 089/930090-11

▸ *Listing 5: Die MAKE-Datei DUMP.MD für die Erzeugung der Debugging-Version.*

```
WL=2
 CC=CL >$*.ERR -c -AM -Gsw -NT DUMP_$* -Od -W$(WL) -Zip $*.C

 DUMP.RES: DUMP.RC DEFS.H DUMP.DLG DUMP.ICO
  RC -r $*

 DUMP.EXE: DUMP.H
  SETDEBUG $*.H OFF

 DUMP.OBJ: DUMP.C DEFS.H DUMP.H
  $(CC)

 PANE.OBJ: PANE.C DEFS.H DUMP.H
  $(CC)

 FILE.OBJ: FILE.C DEFS.H DUMP.H
  $(CC)

 DUMP.EXE: DUMP.OBJ PANE.OBJ FILE.OBJ DUMP.RES DUMP.DFD
  LINK DUMP+PANE+FILE/A:16,/CODEVIEW,,MLIBW+MLIBCEW/NOD,$*.DFD;
  RC $*.RES
```

▸ *Listing 6: Die MAKE-Datei DUMP.MP für die Erzeugung der »Produktion«-Version.*

```
WL=2
CC=CL >$*.ERR -c -AM -Gsw -NT DUMP_$* -Ox -W$(WL) -Zp $*.C

DUMP.RES: DUMP.RC DEFS.H DUMP.DLG DUMP.ICO
 RC -r $*

DUMP.EXE: DUMP.H
 SETDEBUG $*.H OFF

DUMP.OBJ: DUMP.C DEFS.H DUMP.H
 $(CC)

PANE.OBJ: PANE.C DEFS.H DUMP.H
 $(CC)

FILE.OBJ: FILE.C DEFS.H DUMP.H
 $(CC)

DUMP.EXE: DUMP.OBJ PANE.OBJ FILE.OBJ DUMP.RES DUMP.DFP
 LINK DUMP+PANE+FILE/A:16,,,MLIBW+MLIBCEW/NOD,$*.DFP;
 RC $*.RES
```

▸ *Listing 7: Die Ressourcen-Datei DUMP.RC.*

```
/**********************************************************************

 —— D U M P ——————— MS-Windows Application ——————————
                    Resource file

 **********************************************************************

 Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76    (West Germany)

 All rights reserved —— Release 1.00 of 89-Sep-07

 **********************************************************************/

#include <WINDOWS.H>
#include "DEFS.H"

STRINGTABLE BEGIN
  STAPPTITLE,"Datei-Hex-Dump"
  STFILETITLE,"Hex-Dump - Datei »%s«"
  STNOMEM,"Zu wenig Speicher!\012Bitte beenden Sie eine Applikation."
  ST_NO_FILE,"Datei %s nicht gefunden."
  ST_TOO_LONG,"Datei %s ist länger als 64 MBytes."
  ST_BAD_READ,"Von Datei %s kann nicht gelesen werden."
END /* STRINGTABLE */

/* ——————————————
   standard ressources
   ——————————————
*/

ICO_MAIN ICON DUMP.ICO
CRS_SPLITH CURSOR SPLITH.CUR
CRS_SPLITV CURSOR SPLITV.CUR
CRS_SPLITX CURSOR SPLITX.CUR

/* ——————————————————————
   application's menu specification
   ——————————————————————
*/


MNU_MAIN MENU BEGIN
  POPUP "&Datei" BEGIN
    MENUITEM "&Laden...",CMD_OPEN
    MENUITEM SEPARATOR
    MENUITEM "&Schließen",CMD_EXIT
    MENUITEM "&Über...",CMD_ABOUT
    MENUITEM SEPARATOR
    MENUITEM "&Teilen...",CMD_SPLIT,GRAYED
  END
END /* MNU_MAIN */


/* ——————————————————
   dialog boxes specification
   ——————————————————
*/

RCINCLUDE DUMP.DLG
```

▸ *Listing 8: Definitionen der Dialogfelder in der Datei DUMP.DLG.*

```
DB_ABOUT DIALOG 10,10,154,76
CAPTION "Über DUMP"
STYLE WS_CAPTION|WS_POPUP
BEGIN
 CONTROL "Microsoft Windows 2",IDNONE,"static",
         SS_CENTER|WS_GROUP|WS_CHILD,0,5,154,8
 CONTROL "Hex-Dateiinhalts-Anzeige",IDNONE,"static",
         SS_CENTER|WS_CHILD,0,14,154,8
 CONTROL "Version 1.00",IDNONE,"static",SS_CENTER|WS_CHILD,30,34,94,8
 CONTROL "Copyright ; 1989, Marcellus Buchheit",IDNONE,"static",
         SS_CENTER|WS_GROUP|WS_CHILD,0,47,154,9
 CONTROL ICO_MAIN,IDNONE,"static",SS_ICON|WS_CHILD,4,8,36,32
 CONTROL "Ok",IDOK,"button",
         BS_DEFPUSHBUTTON|WS_TABSTOP|WS_GROUP|WS_CHILD,61,59,32,14
END /* DB_ABOUT */

DB_OPEN_FILE DIALOG 10,10,186,114
CAPTION "Dump: Lade Datei"
STYLE WS_CAPTION|WS_POPUP
BEGIN
 CONTROL "",IDFILENAME,"edit",
         ES_AUTOHSCROLL|ES_LEFT|WS_BORDER|WS_TABSTOP|WS_CHILD,
         4,4,178,12
 CONTROL "",IDPATH,"static",SS_LEFT|SS_NOPREFIX,4,20,178,8
 CONTROL "&Dateien:",IDNONE,"static",SS_LEFT,4,32,32,8
 CONTROL "",IDFILELIST,"listbox",
         LBS_STANDARD|WS_TABSTOP|WS_CHILD,4,44,56,56
 CONTROL "&Verzeichnisse:",IDNONE,"static",SS_LEFT,64,32,56,8
 CONTROL "",IDDIRLIST,"listbox",
         LBS_STANDARD|WS_TABSTOP|WS_CHILD,64,44,64,56
 CONTROL "Ok",IDOK,"button",
         BS_DEFPUSHBUTTON|WS_TABSTOP|WS_GROUP|WS_CHILD,132,54,50,14
 CONTROL "Abbrechen",IDCANCEL,"button",
         BS_PUSHBUTTON|WS_TABSTOP|WS_GROUP|WS_CHILD,132,74,50,14
END /* DB_OPEN_FILE */
```

▸▸ *Listing 9: Die Konstanten-Definitionsdatei DEFS.H.*

```
/* menu commands */
#define CMD_OPEN    121
#define CMD_EXIT    122
#define CMD_SPLIT   123
#define CMD_ABOUT   124

/* dialog boxes */
#define DB_ABOUT   500
#define DB_OPEN_FILE 501

/* main menu */
#define MNU_MAIN   900

/* accelerator */
#define ACC_MAIN 900

/* icons */
#define ICO_MAIN 900

/* cursors */
#define CRS_SPLITH 900
#define CRS_SPLITV 901
#define CRS_SPLITX 902

/* strings */
#define STVERSION     0
#define STAPPTITLE    1000
#define STFILETITLE   1001
#define STNOMEM       1002
#define ST_NO_FILE    1500
#define ST_TOO_LONG   1501
#define ST_BAD_READ   1502

/* dialog box entries */
#define IDNONE      0
#define IDTITLE     2000
#define IDVERSION   2001
#define IDICON      2002
#define IDSIGNET    2003
#define IDFLAG      2004
#define IDFILENAME  2010
#define IDFILELIST  2011
#define IDDIRLIST   2012
#define IDPATH      2013
#define IDHSPLIT    2100
#define IDVSPLIT    2101
#define IDPSCROLL   2102 /* ..2105 */
```

▸▸ *Listing 10: Die Linker-Definitionsdatei DUMP.DFD für das Debugging.*

```
NAME DUMP
REALMODE
EXETYPE WINDOWS
DESCRIPTION 'Hex file dump program'
STUB 'WINSTUB.EXE'
CODE MOVABLE PRELOAD
DATA MOVABLE MULTIPLE
HEAPSIZE  4096
STACKSIZE 4096

SEGMENTS
 DUMP_DUMP FIXED PRELOAD
 DUMP_PANE FIXED PRELOAD
 DUMP_FILE FIXED PRELOAD

EXPORTS
 fwMain @1
 fwSplit @2
 fdAbout @3
 fdOpenFile @4
```

▸▸ *Listing 11: Die Linker-Definitionsdatei DUMP.DFP für die »Produktion«.*

```
NAME DUMP
REALMODE
EXETYPE WINDOWS
DESCRIPTION 'Hex file dump program'
STUB 'WINSTUB.EXE'
CODE MOVABLE DISCARDABLE
DATA MOVABLE MULTIPLE
HEAPSIZE  4096
STACKSIZE 4096

SEGMENTS
 DUMP_DUMP MOVABLE LOADONCALL DISCARDABLE
 DUMP_PANE MOVABLE LOADONCALL DISCARDABLE
 DUMP_FILE MOVABLE LOADONCALL DISCARDABLE

EXPORTS
 fwMain @1
 fwSplit @2
 fdAbout @3
 fdOpenFile @4
```

Produktübersicht:

Programme für Microsoft Windows

Windows hat in letzter Zeit stark an Bedeutung gewonnen und auch die Akzeptanz unter Anwendungsentwicklern hat zugenommen. Dies ist vor allem an der Vielzahl der Softwareprodukte zu erkennen, die unter Microsoft Windows laufen. Auf den folgenden Seiten bringen wir eine Übersicht der wichtigsten Programme für Microsoft Windows. Für jedes Produkt ist aufgeführt: der Produktname, eine Kurzbeschreibung, der Name des Herstellers und/oder anderer Bezugsmöglichkeiten.

Wenn Sie selbst ein interessantes Windows-Produkt entwickelt haben oder kennen und möchten, daß es in die nächste Windows-Produktübersicht aufgenommen werden soll, schreiben Sie an:

Microsoft GmbH, Christian Wildfeuer, Edisonstr. 1, 8044 Unterschleißheim

Datenbank

Beschreibung	Hersteller/Vertrieb	Produktname
Erstellung von MS-Windows-Programmen, die Dateien im dBase-Format verwalten und auswerten. Leistungsumfang: 1. Dateiverwaltungsfunktionen 2. Umsetzung ASCII-/ANSI-Zeichensatz 3. Anpassung der dBase-Datenstrukturen und Datentypen an die »C«-Konventionen 4. Generierung der Leer- und Include-Dateien 5. Zentrale Pflege der Feldtitel und der feldbezogenen Plausibilitätsbereiche für die Windowsmasken. 6. Maskenorientierte Such- u. Selektionssprache nach der Philosophie »Query by Example«: Bool'sche Operatoren, vollständige AOS-Logik, arithmetische und alphabetische Vergleiche, Mustererkennungsfunktionen.		**DBF-Link / DBF-Query**
OMNIS Quartz ist ein Datenbanksystem, das alle Vorteile von MS-Windows nutzt. Ohne sich Kenntnisse der Windows-Programmierschnittstelle aneignen zu müssen, kann der Anwendungsentwickler Applikationen schreiben, die von Dialogboxen über Pull-down-Menüs bis hin zu dynamischen Datenaustausch, alle Windows-Konzepte voll nutzen. Erhältlich als Single- und Multi-User-Version. Die Leistungsfähigkeit in Zahlen: • Unbegrenzte Anzahl von Datensätzen in jeder Datei • 120 Felder pro Eintrag • Maximal 288.000 Zeichen pro Eintrag • 60 Tabellen pro Datenbank (Look-up Tabellen) • Überprüfung der Eingabe auf Kriterien • Bis zu 12 Index-Felder • 60 Dateien gleichzeitig geöffnet • Datenaustausch per DIF, dBase, SYLK, ASCII, WK1, WKS-Dateien • Reportgenerator mit WYSIWYG-Darstellung aller Schriften (MS Windows-Fonts) in verschiedenen Schriftgrößen und Typen	BLYTH Software Ltd. Mitford House, Benhall GB Saxmundham, Suffolk IP17 1JS Tel.: 0045/0728/3011 Fax: 0045/0728/4154 Mikro Partner GmbH, 2000 Hamburg 1 Intellis, CH - Ebmatingen	**OMNIS Quartz**
Sie können mit Ontrak, dem Information Organizer, eine umfassende Menge von Informationen schnell und einfach verwalten. Nutzen Sie OnTrak, um Kundenlisten, Werbematerial, Gewinne, Aufträge, Geschäftskontakte und vieles mehr zu verwalten. OnTrak bietet ein bekanntes Rolodexähnliches Interface, welches das Sortieren und Sichten Ihrer Daten wesentlich vereinfacht. Sie können eine unbegrenzte Anzahl von Karteikarten erstellen, um jede Art von Information zu verwalten. Sie könen OnTrak so modifizieren, daß es die Art von Informationen bearbeitet die wichtig für Sie sind. Sie können pro Karte bis zu acht Seiten an Detail-Informationen eintragen! OnTrak hat die einzigartige Möglichkeit Ihre Informationen nach jedem beliebige Feld auf Ihrer Karte zu sortieren. OnTrak ist schnell - Ihre gesamte Kartei läßt sich in einem Bruchteil von Sekunden sortieren, so haben Sie immer sofortigen Zugriff auf auf Ihre Daten.	Active Software Corporation 1208 Apollo Way, Suite 507 Sunnyvale, CA 94086 USA	**OnTrak**
Winbase ist ein Datenbanksystem, das sowohl als eigenständige Datenbank, als auch als Editor für dBase-kompatible Dateien eingesetzt werden kann. Vorteile: Bequeme Bedienung per Maus und Austausch von Daten zwischen verschiedenen Applikationen über das Clipboard. Außerdem kann man mehrere Fenster gleichzeitig öffnen und zwischen den Datenbank-Dateien Informationen transferieren. Eigenschaften: • mehrere Fenster gleichzeitig • horizontales und vertikales Scrollen • Sortierung der Datensätze (aufsteigend/absteigend nach einem oder mehreren Feldern) • Suche nach Wörtern, Zahlen Ausdrücken (wie bei Textverarbeitung) • Berechnung von Summen, Durchschnitten, Minima und Maxima findet fast von ganz allein statt • Ausgabe von Datensätzen erfolgt wahlweise auf den Drucker, in eine Datei oder auf das Window-Clipboard, von dem aus die Daten in eine weitere Applikation übernommen werden können.	Pioneer Software Markt & Technik Verlag AG, 8013 Haar	**WinBase**
dBase-kompatible Datenbank. Speichert Text und Grafik. Maximale Anzahl der Einträge: 32.000.	Softline GmbH	**WindowsFiler**

Tabellenkalkulation/Finanzen

Beschreibung	Hersteller/Vertrieb	Produktname
BLUES bringt Ihnen aktuelle Marktdaten in Ihr eigenes Spreadsheet. BLUES ist ein Verbindungswerkzeug (Connectivity Tool) - eine Brücke zwischen aktuellen Marktdaten und Microsoft Excel. Mit BLUES lassen sich mehrere Fenster gleichzeitig mit allen Informationen darstellen, die auf RDX II verfügbar sind (u.a. Reuters, Telerate und die Stock Eschange Computer Readable Services. Ein Microsoft Excel-Worksheet empfängt mit Hilfe von BLUES automatisch die neuesten Daten uns macht Neuberechnungen und Updates aller damit verbundenen Grafiken möglich. Gestalten Sie in Excel Ihre eigenen Portfolios, Risikomodelle, Optionsstrategien, etc. BLUES wird diese Modelle mit lebenden Daten füllen. BLUES kann gleichzeitig Seiteninformation, Telexe und Teilseiten-Information übertragen. Zur selben Zeit können diese Informationen gespeichert werden. BLUES erweitert die Möglichkeiten des RDX II Trading Room beträchtlich. Die Datenübertragung erfolgt über RDX II Network, das von IBM vertrieben wird.		**BLUES**

Produktname	Beschreibung	Hersteller/Vertrieb
CAS Costing	Das FDS-Costing soll zur Unterstützung der Kalkulation von Schuh- oder Bekleidungsteilen herangezogen werden und das bestehende FDS- bzw. ADS-CAD-System in betriebswirtschaftlicher Hinsicht ergänzen. Dazu können die innerhalb des CAD-Systems bereits erfaßten, berechneten und in der CUT- oder PLOT-File gespeicherten Daten wiederverwendet werden.	CAS GmbH EKZ Am Alten Markt Hauptstr. 46 6780 Pirmasens
CFO Advisor	CFO Advisor ist mehr als ein Spreadsheet. Es liefert dem Manager sofort leistungsfähige Entscheidungshilfe durch Auswertung von Daten aus dem Rechnungswesen, um dadurch die gegenwärtige Position Ihres Unternehmens einzuschätzen. CFO Advisor hilft Ihnen, den Einfluß verschiedener Strategien vorherzusagen und Leistungsstandards zu bestimmen, um gewünschte Ziele zu erreichen. Sein einzigartiger Blueprint-Bildschirm identifiziert die Zusammenhänge zwischen den wichtigsten Leistungsgebieten des Unternehmens, auf die das Management direkten Einfluß nehmen kann, und auf die wichtigsten Ergebnisbereiche, die die Leistungsfähigkeit, Effizienz und Profitabilität des Unternehmens messen. CFO Advisor ermöglicht Ihnen direkten Input und bietet Schnittstellen für General Ledger Systeme wie ACCPAC Plus, Solomon III, RealWorld, und CYMA. Ebenso verfügt es über Schnittstellen für Lotus 1-2-3 und SuperCalc.Zudem ist ein dynamischer Datenaustausch mit Microsoft Excel möglich.	Financial Feasabilities, Inc. 9454 Wilshire Blvd./Penthouse Suite USA Beverly Hills, CA 90212 Tel.: 001/213/2788000
COMPETE! 3.01	Compete! lets users track five variables - typically financial results, products, markets, time periods and competitors - rather than the standard two in programs such as the Lotus 1-2-3 spreadsheet. An analyst can easily spot numbers that may indicate problems, or perform »what-if« analysis to see how a specific change would ripple through the business. For instance, an airline could experiment to determine the optimal fare cut for a particular type of ticket that is generating low revenue at a certain time of the year in a few markets, taking into account how competitors generally have responded. While this sort of analysis is done now, it typically takes many hours to poroduce the sort of result that Compete! can produce in seconds. »You put your fingers on all these (varibables) mentally, but pretty soon you run out of fingers and you have to stop and start all over... Not with Compete!	
Excel	Microsoft Excel vereint Tabellenkalkulation, die Leistungsfähigkeit einer schnellen Datenbankfunktion sowie außergewöhnlich leistungsfähige Geschäftsgrafikfunktionen zu einer leistungsstarken Lösung. Stellen Sie mehrere Arbeitsblätter und Grafiken gleichzeitig am Bildschirm dar, konsolidieren Sie Informationen, erstellen Sie Reports und verknüpfen Sie Arbeitsblätter, so daß sich Änderungen in allen verbundenen Bereichen vollziehen - all das ist mit Microsoft Excel problemlos möglich. Aufgrund seiner richtungsweisenden grafischen Bedieneroberfläche ist Microsoft Excel äußerst leicht und schnell zu erlernen. Ein didaktisch strukturiertes Lernprogramm führt Sie Schritt für Schritt durch alle Funktionsbereiche. Häufiger benötigte Arbeiten können Sie schnell und einfach automatisieren. Makros lassen sich vom Makrorekorder automatisch aufzeichnen oder direkt in der Makrosprache bis hin zu eigenständigen Anwendungen erstellen.	Microsoft GmbH Edisonstr. 1 8044 Unterschleißheim ACCESS Computer Vertriebs GmbH ALSO ABC Trading AG BSP Softwaredistribution GmbH Cfm Computertechnik AG Computer 2000 AG Softexpress Datenverarbeitungs- und Betriebsberatungs- ges.m.b.H.
NEXPERT OBJECT	Nexpert Object bietet eine Umgebung für die Entwicklung und den Ablauf von Expertensystemen in der Diagnostik, Finanz, Simulation, Planung, Ablaufkontrolle und mehr. Dieses hybride regel- und objektorientierte System läuft auf dem IBM PC/AT, PS/2 und anderen auf dem Intel 80286/80386 Chip basierenden Rechnern unter MS-DOS mit Microsoft Windows wie auch unter OS/2. Nexpert ist vollständig in C programmiert, um ein Höchstmaß an Performance und Integration zu gewährleisten. Es kann durch eine AI-Bibliothek voll in bestehende Betriebsumgebungen eingebunden werden. Nexpert unterstützt unter Microsoft Windows die Protokolle des Dynamischen Daten Austausches und der Dynamic Link Bibliotheken. Die umfassende Grafikschnittstelle von Nexpert ermöglicht Ihnen, Regeln zu editieren sowie Kontrollstrukturen aufzubauen. Seine offene ereignisgesteuerte Architektur erlaubt die Entwicklung von Echtzeit- und On-Line-Anwendungen und unterstützt voll die Kommunikation mit Standard-Datenbanken.	Neuron Data 444 High Street USA Palo Alto, CA 94301
Option Risk Management	Option Risk Management ist ein Analyse-Programm für den professionellen Broker ebenso wie für den privaten Anleger. Sekundenschnell lassen sich Prämien, Delta/Vega/Gamma, Volatilität sowie Gewinn/Verlust Potential auf Optionspositionen inkl. »Underlying« aller Märkte berechnen. Die Daten können direkt von Kommunikationssystemen oder internen Großcomputern eingelesen werden. ORM wird erfolgreich von namhaften Brokern an den amerikanischen Futures- und Optionsmärkten sowie von Schweizer Banken für den SOFFEX- und OTC-Optionshandel eingesetzt. Als Ergänzung ist ab ca. Sommer '89 auch Real-Tick III als »Real-time-quote-screen und Grafikprogramm« erhältlich.	Townsend Analytics Ltd. 30 South Wacker Drive/Suite 1117 USA Chicago; IL 60606 TOFF Consulting & Finanz AG

Grafik

Produktname	Beschreibung	Hersteller/Vertrieb
Arts & Letters	Mit Arts & Letters (TM) erstellen Sie auf professionelle Art präsentationsfähige Diagramme, Illustrationen, Organigramme, Broschüren und andere Grafiken. Mit Arts & Letters erzielt man auch ohne Zeichentalent schöne und überzeugende Ergebnisse. Schneiden, kopieren oder malen Sie Ihre Kompositionen in bekannte Desktop-Publishing-Anwendungen wie z.B. Aldus Pagemaker und Ventura Publisher. Wählen Sie aus über 3700 professionell gezeichneten Clip-Art Bildern und 15 High-Resolution-Schriftarten. Tausende Bild-Optionen und Logos sind ebenfalls verfügbar, ebenso viele ausgefallene Schriftarten. Oder benutzen Sie den Graphics Editor, um eigene Ideen zu verwirklichen, bereits über 1000 im System vorhandene Symbole zu verändern oder um per Scanner eingefügte Bilder zu verarbeiten. Schrifttypen und Symbole können ohne Beeinträchtigung der Bildqualität verkleinert bzw. vergrößert, gedreht, verbreitert, verschmälert, etc. werden. Jeder Laserdrucker ermöglicht den Ausdruck in Präsentationsqualität. Text und Grafik aus anderen Windows-Applikationen sind übertragbar.	Computer Support Corporation 15926 Midway Road USA Dallas, TX 75244 Tel.: 001/214/661 896 Softline Company, 7602 Oberkirch
ClickArt EPS	Click Art EPS Illustrations ist ideal zur Vervollständigung von allen Publikationen, an die höchste Ansprüche gestellt werden. Das Programm enthält Symbole und Bilder u.a. aus den Gebieten Industrie und Technik, Landkarten, Musikinstrumente, Verkehrswesen, Jahreszeiten, Sport, Gesichter und Menschen, Bürogeräte, Nahrungsmittel und Getränke, etc. EPS Illustrations ist kompatibel mit PC PageMaker 2.0 und anderen EPS-lesenden Produkten.	T/Maker Company 1390 Villa Street/Mountain View USA Beverly Hills, CA 90212 Tel.: 001/415/9620195 Fax: 001/415/9620201 CCP Software, 3550 Marburg
ClickArt Image Portfolios	Das sechsteilige Programm enthält Bilder und Symbole aus folgenden Bereichen: • ClickArt EPS Business Art • ClickArt Business Images• ClickArt Publications • ClickArt Personal Graphics • ClickArt Holidays • ClickArt Christian Images	T/Maker Company 1390 Villa Street/Mountain View USA Beverly Hills, CA 90212 Tel.: 001/415/9620195 Fax: 001/415/9620201 CCP Software, 3550 Marburg

Beschreibung	Hersteller/Vertrieb	Produktname
ClickArt Scrapbook+ ist ein intelligenter Grafik- und Textmanager, mit dem Sie auf zwei Arten Grafik und Text speichern und ordnen können: durch Mouse-Steuerung oder Menüwahl. Sie finden leicht das gesuchte Bild und können es sofort in PC PageMaker, Microsoft Windows Write oder jeder anderen Windows-Applikation einbinden. ClickArt Scrapbook+ ist leicht zu bedienen und in hohem Maße anschaulich. Es fertigt automatisch Miniaturkopien Ihrer Entwürfe an und speichert diese in eine Datei. Dabei können Sie einen visuellen Index aller Ihrer Bilder auf einen Blick sehen. Außerdem ist es möglich, Bilder mit Schlüsselworten zu benennen, um sie später blitzschnell per Suchfunktion wiederzufinden. ClickArt Scrapbook+ ermöglicht Ihnen ebenfalls die Speicherung und Ordnung von Text-Files, Excel-Daten (SYLK-Dateien), Excel-Grafik, EPS-Dateien, Meta-Files etc. Voraussetzung dafür: Microsoft Windows 2.0 oder höher.	T/Maker Company 1390 Villa Street/Mountain View USA Beverly Hills, CA 90212 Tel.: 001/415/9620195 Fax: 001/415/9620201 CCP Software, 3550 Marburg	**ClickArt Scrapbook+**
Clip Art 3-D bietet CD-ROM-Anwendern jetzt Zugriff auf mehr als 2500 dreidimensionale Einzelbilder (Vektorgrafik) für Desktop Publishing, Design und Präsentationsgrafik. Auf nur einer Disc stehen Ihnen Darstellungen zu zahlreichen Themenkreisen zur Verfügung: Menschen, Geografie, Transportwesen, Architektur, Nahrungsmittel, Maschinen, etc. Außerdem eine Reihe von 3-D-Schriften. Alle Einzelbilder lassen sich individuell verändern: in Perspektiven, Lage und Größe. Über acht Lichtquellen könne Ausleuchtung und Farbgebung jederzeit verändert werden. Das Programm enthält 500 bereits zusammengesetzte Abbildungen. Sie können selbstverständlich auch Ihre eigenen Bildkompositionen erstellen. Ihrer Kreativität sind dabei keine Grenzen gesetzt. Ausgewählte und nach Ihren Wünschen editierte Bilder lassen sich im PIC-, WMF- oder PostScript-Grafikformat abspeichern. Damit werden Layoutgestaltung und Design am PC so einfach wie noch nie. Erhältlich für PC und für Apple Macintosh.		**Clip Art 3-D**
Mit COREL DRAW können Sie jeden Text verändern, drehen, vergrößern, verkleinern, dehnen, spiegeln und an jede beliebige Linie anpassen sowie verschiedene Abstände zwischen den einzelnen Buchstaben eines Wortes frei wählen - und dies alles in WYSIWYG. Es stehen Ihnen über 57 Schriften in beliebiger Größe wie auch mehrere Schönschreib-Schriftarten zur Textgestaltung zur Verfügung. Mit COREL DRAW ist es äußerst einfach, Objekte zu bearbeiten. Klicken Sie irgend einen Teil des Objektes an, verändern Sie diesen frei nach Ihren Wünschen, duplizieren Sie Abbildungen oder speichern Sie häufiger benötigte Arbeitsschritte als leistungsstarke Makros ab. COREL DRAW stellt Ihnen mehrere Bibliotheken von Click-Art-Abbildungen zur Verfügung, die Sie über Dialogboxen ganz einfach in Ihr Arbeitsblatt übernehmen können. COREL DRAW ist der Partner für DTP und Textverarbeitungsprogramme. Sie können TIFF-, PCX-, PIC- und Adobe Illustrator-Dateien importieren. Die deutsche Produktversion erscheint im 3. Quartal '89.	COREL Systems Corporation 1600 Carling Avenue CAN Ottawa, Ontario K1Z 8R7 EDTZ DTP Partner Jakobi Systemhaus	**COREL DRAW!**
Design/OA ist eine offene Architektur, die Funktionen zur Erzeugung von interaktiven, grafischen Anwendungsprogrammen zur Verfügung stellt. Insbesondere wird die Implementierung von grafischen Anwendungen unterstützt, denen eine logische Verbindung von Objekten zugrunde liegt, wie z.B. Petri-Netz-Editoren oder grafischen Simulatoren. Design/OA bietet eine vollständige horizontale Programmierumgebung. Dem Programmierer stehen alle Funktionen von Design/2.0 zur Verfügung. Daher kann er sich völlig auf das eigentliche Problem, die zu implementierende Methode, konzentrieren. Der Programmierer bestimmt die Art und Anzahl der Sorten von Objekten, die Regeln für die Verbindung von Objekten sowie die Semantik für die Beziehungen zwischen den Objekten.	Meta Software Corporation 150 Cambridge Park Drive USA Cambridge, MA 02140 C.I.T. GmbH NOSER AG	**Design/OA**
Desktop Publishing für Werbebüros, Grafiker, technische Zeichner, Typographen, Architekten, Designer. Diagraph Windows ist ein komplettes WYSIWYG-DTP-Programm für Drucksachen aller Art. Diagraph Windows ist die ideale Ergänzung zu den herkömmlichen Grafik- und Publishing-Programmen (z.B. PageMaker, Ventura, Office Publisher, First Impression, Lotus, Havard Graphics u.a.). Spezialeffekte: Texte sind frei drehbar, lassen sich rastern, als Outlines verwenden, stauchen und dehnen, etc. Eingebaute Funktionen: • Textimport über ASCII aus Textverarbeitungen • Mehrspaltiger Umbruch (unbegrenzte Seitenzahl) • Schriften von 2-2000 Punkt in 1/10 Punktschritten • Grafikimport über Scanner/Grafiksoftware (TIFF/PIC) • 45 Schrifttypen (15 im Grundpaket; weitere optional) • Manuelle Spationierung • Grafikfunktionen (z.B. Kreise, Linien, Quadrate) • Zoom (Lupe) und Ganzseitenansicht • Raster für genaue Ausrichtung einblendbar • für Nadeldrucker (9-24 Nadeln) und Laserdrucker • Ausgabe auf Satzbelichter über Postscript (max. 2540 dpi) • Farbseparation und Farb-Postscript-Ausgabe (optional) • Farbdruck auf Farbdruckern (z.B. HP PaintJet, Nec CP5/6) • 3300 Grafiken im Lieferumfang enthalten	Softline Company, 7602 Oberkirch	**Diagraph Windows**
Zusatzprogramm für Diagraph Windows. Kurvenfunktionen (Bezierkurven) für Freihandzeichnen und komplizierte Grafik in Vektorqualität, freie Editierbarkeit aller Symbole und Vektorgrafiken; Übernahme gescannter Strichvorlagen und Halbtonfotos zur Weiterverarbeitung als Vektoren (über TIFF-Grafikimport); Vektorisierung von Firmenlogos, Signets, Unterschriften, u.a. für maximale Ausgabequalität (auch nach Vergrößerung und Drehung) sind Erweiterungen für Diagraph Windows. Bei gescannten Logos in Pixelgrafik (z.B. 300 dpi oder weniger) kann eine Vergrößerung oder Drehung des Logos mit herkömmlichen Programmen nur mit Qualitätseinbußen und Sägezähnchen erreicht werden. Nach einer Vektorisierung im Editor wird immer die maximale, randscharfe Qualität erreicht. Alle in Diagraph Windows und im Editor erstellten Spezialeffekte, Texte, Seiten, Grafiken oder Logos können problemlos mit anderen Programmen ausgetauscht werden. Als Ausgabeformate dienen CGM (Computer Graphics Metafiles) EPS (Encapsulated Postscript) und WMF (Windows Metafiles). Diagraph Windows arbeitet mit CGA, Hercules, EGA, VGA und Ganzseitenschirmen bis 20 Zoll.	Softline Company, 7602 Oberkirch	**Diagraph Windows Editor**
Digital Photolab erlaubt das Retuschieren und Editieren von Halbtongrafiken, die aus Scannern oder als TIFF-Formate eingelesen wurden. Empfohlene Konfiguration: VGA-Karte und Postscript-Drucker.	Softline Company, 7602 Oberkirch	**Digital Photolab**
Anwendungsbereiche: für grafische Repräsentationsmöglichkeiten quantitativer und logischer Informationen. Die Anwendung DrawCel versetzt Excel in die Lage, einfache geometrische Elemente wie Linien, Polygone, Kreise, Intensitätspfeile und Rechtecke darzustellen. Dabei wird jedes dieser Elemente durch seine Parameter (X-Position-, Y-Position, Breite, Höhe, Radius, etc.) bestimmt. Im Gegensatz zu herkömmlichen Windows-Zeichenprogrammen sind die Parameter dieser Elemente mit den Kalkulationswerten beliebiger Excel-Tabellen verknüpfbar. Dies bedeutet, daß sich bei Veränderung eines dieser Tabellenwerte sofort die Zeichnung an die neuen Parameterwerte anpaßt (= dynamisches Zeichnen). Neben den genannten geometrischen Grundelementen ist außerdem ein Zeichensatz ansprechbar, der sich unabhängig von den verknüpften Excel-Tabellen versetzten, vergrößern, verkleinern und rotieren läßt. DrawCel enthält einen menügesteuerten Funktionsbereich zum Anlegen, Verwalten und Editieren von Zeichnungsdateien. Dabei ist die Einrichtung von bis zu 253 Zeichnungsebenen (Layers) möglich.	K. Raue / Management Systeme GmbH Bunsenstr. 22 6100 Darmstadt Tel.: 06151/82077	**DrawCel**

Produktname	Beschreibung	Hersteller/Vertrieb
Dynamisches Zeichnen	Anspruchsvolle Geschäftsgrafiken (z.B. Portfolio Darstellungen, Struktugramme, Terminpläne) und technische Präsentationsgrafik (z.B. Material- und Stofflußpläne auf der Grundlage von Excel Tabellen). Dynamisches Zeichnen erlaubt es, Zeichnungen in direkter Abhängigkeit von Excel-Tabellen zu erzeugen. Dabei lassen sich Merkmale von Geschäftsgrafiken mit Möglichkeiten einfacher CAD-Programme kombinieren. Durch die Möglichkeit, Bezüge durch Formeln zu verknüpfen, können komplexe Zeichnungen erstellt werden, die bei Änderung der Parameter in der zugrundeliegenden Tabelle neu berechnet und dargestellt werden. Die farbigen Zeichnungen können auf der Grundlage folgender geometrischer Elemente erzeugt werden: Linien, Polygone, Rechtecke (rotierbar), Kreise und Kreisbögen, Ellipsen und Ellipsenbögen, Pfeile, rotierbarer Text. Der Benutzer kann sich eigene Symbolbibliotheken (z.B. Landkarten, technische Symbole) erstellen, so daß die Definition von ähnlichen Grafiken sehr effizient durchzuführen ist. Die Zeichnungen lassen sich auf bis zu 256 Layern definieren. Excel läßt sich dabei uneingeschränkt weiternutzen.	
Graph Plus	Noch nie war es so einfach, Zahlenkolonnen in zwei- und dreidimensionale Geschäftsgrafik umzusetzen. Ohne großen Aufwand kann man mehrere Diagramme in ein attraktives Umfeld setzen, z.B. Säulendiagramme der Bevölkerung in einen Aufriß einer Europakarte. Eine Vielzahl von Hintergründen und illustrativen Symbolen ist integriert. Für eigene Entwürfe und den Einbau von Firmenlogos stehen Vektorgrafik-Funktionen zur Verfügung. Zusätzlich wird eine Bibliothek mit tausend fertigen Illustrationen angeboten. Alle nur denkbaren Darstellungsformen sind möglich: Balken, Säulen, Linien, Kuchen, Kurven, Verteilungen (Scatter), Tabellen u.v.m. Daten können in Graph Plus aufbereitet werden oder direkt aus Lotus 1-2-3, VisiCalc, Multiplan, Excel, Symphony u.a. übernommen werden. Zur Beschriftung gibt es die üblichen Satzschriften (z.B. Times und Helvetica), hochwertige Ausgabe auf Linotronic, Laserdrucker (mit/ohne Postscript) und Plotter. Special Feature: Dynamic Data Exchange. Wenn in einer Datentabelle, auch außerhalb von Graph Plus ein Wert geändert wird, korrigiert das Programm automatisch die dazu gehörige Grafik.	Micrographix, Inc. 1820 North Greenville Ave. USA Richardson, TX 75081 Tel.: 001/214/2341769 Fax: 089/93006011 Softline Company, 7602 Oberkirch CfM Computertechnik A.G. - CH
Image Folio	Mit der neuen CD-ROM-Software Image Folio haben Anwender von CD-ROM-Laufwerken jetzt Zugriff auf über 4000 Realbilder für den Einsatz in Desktop Publishing, Design und Grafikpräsentationen. Alle auf der CD verfügbaren Bilder lassen sich Ihren Vorstellungen entsprechend verändern. Image Folio bietet dazu eine Vielzahl an Retuschiertechniken wie Airbrush, Bildfreistellung und -beschriftung. Sogar die Helligkeits- und Farbwerte eines Bildes können variiert werden. Um das für Ihren Zweck geeignete Bild zu finden, geben Sie einfach ein passendes Suchwort ein. Die Bildsuche über dieses Schlüsselwort liefert Ihnen schnell die Auswahl an geeigneten Bildern. Falls Sie ein ausgewähltes und nach Ihren Wünschen editiertes Bild weiterverwenden wollen, so können Sie es mühelos im TIFF- oder TGA-Format abspeichern. Damit werden Bild- und Layoutgestaltung am PC so einfach wie noch nie. Erhältlich für PC und für Apple Macintosh.	
MetaDesign	Das Werkzeug für Grafik und Text zur Visualisierung komplexer Systeme. MetaDesign ermöglicht ein schnelles Erstellen von objektorientierten Grafiken mit integriertem Text. Damit ist MetaDesign ideal geeignet, um komplexe Systeme zu analysieren, zu organisieren und zu dokumentieren. MetaDesign zeichnet sich durch die Fähigkeit zur Darstellung strukturierter Grafik aus. Logische Verbindungen von Objekten bleiben auch nach Verfeinerung und Änderung der Position eines Objektes erhalten. Dokumente sind in Seiten organisiert, die hierarchisch angeordnet werden können. Durch Vergröberungs- und Verfeinerungsmechanismen können die Seiten übersichtlich gestaltet werden, ohne die logischen Beziehungen der Objekte untereinander zu verändern. Eine unbegrenzte Anzahl von Objekten und Objektkombinationen kann erzeugt und in einer anwendungsspezifischen Palette zur Verfügung gestellt werden. Zu jedem grafischen Objekt kann gestalteter Text erfaßt werden. Texte und Grafiken aus Hypertext-Verbindungen können logische Beziehungen zwischen Texten seitenübergreifend herstellen.	Meta Software Corporation 150 Cambridge Park Drive USA Cambridge, MA 02140 C.I.T. GmbH, 1000 Berlin NOSER AG, CH - 8403 Winterthur
Micrographix Designer	Zeichen- und Illustrationssoftware für die Aufbereitung von AutoCAD-Zeichnungen für die Übernahme in DTP-Programme (PageMaker, Ventura). Aus den Strichzeichnungen einer importierten AutoCAD-Zeichnung lassen sich perfekte farbige und gerasterte Illustrationen einlesen.	siehe Graph Plus
NoVA-3D	Das Programmpaket für visuelle Gestalter: Grafiker, Produktdesigner, Innenarchitekten, Architekten, Zeichner sowie alle AutoCAD-Anwender. NoVA-3D ist das erste Desktop 3D-Programm sowie das erste professionelle 3D-System mit dem unter Windows/386 funktionierenden Multitasking. NoVA-3D ist kompatibel zu Ventura, PageMaker, usw. (PC/MS-DOS und Apple Macintosh). NoVA-3D verarbeitet 3-dimensionale Bilder farbig, mit echtem Schattenwurf, Eingabe von Standort und Tageszeit möglich. Ermöglicht das Durchschreiten von Modellen.	
Paintbrush Plus	Das meistverkaufte PC-Malprogramm PC Paintbrush gibt es auch in einer Windows-Version. Von den anderen Mitgliedern der Paintbrush-Familie unterscheidet sich Paintbrush+ vor allem durch das Scanner-Modul. Es ermöglicht einen sogenannten Low-Level-Prescan. Das heißt, eine Vorlage wird zunächst mit einer niedrigen Auflösung grob, aber schnell eingescannt. Der Benutzer entscheidet aufgrund dieses »Prescans«, welchen Ausschnitt der Vorlage er endgültig haben möchte. Der so markierte Ausschnitt wird dann mit der höchstmöglichen Auflösung endgültig gescannt und kann weiterverwertet werden. Durch dieses Verfahren wird deutlich Zeit gespart.	ZSoft Corp. 450 Franklin Road USA Marietta, GA 30067 Tel.: 001/404/4280008
Pixie	Pixie ist ein einfach zu erlernendes Präsentationgrafik-Paket. Sie Erstellen und Verändern mit direktem Zugriff, ohne umständliche Kommandos. Bei Tortendiagrammen ziehen Sie beispielsweise ein Stück ganz einfach per Maus-Steuerung heraus, anstatt umständliche Befehle über Menü oder Tastatur eingeben zu müssen. Die breite Spanne an Zeichen-Tools (ähnlich wie beim Aldus PageMaker) ermöglicht es, die Charts zu illustrieren oder Freihandzeichnungen aus dem Stegreif zu erstellen. Pixie bietet außerdem einen sehr hohen Standard in der Farbdarstellung. Zusammen mit einem Farbmonitor und einer digitalen Kamera ist die Pixie-Präsentation kaum noch zu überbieten. Aus Lotus 1-2-3 Spreadsheets können benannte Bereiche und Grafiken übernommen werden. Über das Windows Clipboard kann man Windows-Bilder, Bitmaps oder Text innerhalb der Microsoft Windows-Umgebung portieren. Sogar Image-Dateien aus folgenden nicht unter Windows laufenden Applikationen können übernommen werden: CGM Industriestandard, VideoShow und Zenographics Mirage Image-Dateien. Außerdem können die mit Pixie erstellten Bilder in CGM, Mirage und Matrix-Kameras übertragen werden.	Zenographics 19752 MacArthur Blvd./Suite 250 USA Irvine, CA 92715 Tel.: 001/714/8516352 EGG's, Computer 2000 AG, 8000 München 70
Pro3D	Für Architektur, Innenarchitektur, Konstruktion, Electronic Publishing, Forschung, Lehre. Pro3D ist ein Windows-Grafikpaket für zwei- und dreidimensionale Zeichenaufgaben. Körper und Flächen lassen sich maßstabsgetreu mit der Maus konstruieren. Küchenstudios, Sanitärbetriebe, Innenarchitekten erstellen nach Wünschen Ihrer Kunden Pläne und perspektivische Darstellungen von Räumen, die eingerichtet werden sollen. Möbelstücke, Fenster, Badewannen u.ä. lassen sich mit einem einzigen Befehl aus dem »Teilelager« abrufen. Bauteile wie etwa die Kurbelwelle eines Benzinmotors können ebenso wie der Entwurf einer Verpackung schnell und einfach maßstabsgetreu gezeichnet werden. In Forschung und Lehre können komplexe Körper und Strukturen anschaulich dargestellt und von allen Seiten betrachtet werden.	Softline Company, 7602 Oberkirch

Beschreibung	Hersteller/Vertrieb	Produktname
Präsentationsgrafik vom Textkonzept bis zur fertigen Präsentation. Marketing, Managementpräsentationen, Vorträge, Schulung. Alle Bereiche einer Präsentation werden von XEROX PRESENTS abgedeckt: Grobentwurf, Konzept, Texterfassung- und -gestaltung, Zahlendiagramme, Ausgabe von Dias, Farbhellraumfolien für Vorträge sowie Sprechernotizen und Handouts für Zuschauer. Unterstützt professionelle Ausgabegeräte. Grafiken mit 3D-Effekt, vordefinierbare Diagrammformate, verschiedene Zeichenwerkzeuge, Cut- und Pastefunktion, BitStream-Schriften einsetzbar, mit anderen Windowsapplikationen kompatibel. Dateneingabe via Clipboard oder Metafile, TIFF, EPS, PCX, Lotus Freelance und Zenographics Pixie-CGM-Dateien.	Cricket Software, Inc. Computer Graphix AG, CH - 8620 Wetzikon	**XEROX PRESENTS**

Kommunikation

Beschreibung	Hersteller/Vertrieb	Produktname
Da Vinci eMAIL ist das erste Electronic Mail Paket für Microsoft Windows - und wurde vom amerikanischen LAN-Magazine zum Produkt des Jahres gewählt. Sie können Nachrichten, Files und den Inhalt des Windows Clipboards an andere Benutzer eines lokalen Netzwerks schicken. Bei eingehenden Nachrichten gibt der Bildschirm wahlweise ein optisches oder akustisches Signal oder beides. Der Message Editor unterstützt die normalen Kopierfunktionen von Windows und ermöglicht es, Files aus dem Speicher zu laden. Da Vinci eMAIL läuft auch auf NetBIOS. Seine modulare Netzwerk-Architektur unterstützt DECNET, UUCP, X.400 und andere Netzwerke. Dynamischer Datenaustausch (DDE) zwischen einzelnen Windows-Programmen ist möglich.	Da Vinci Systems Corporation Tel.: 001/919/8392000 Softline Company, 7602 Oberkirch	**Da Vinci eMAIL**
Intalk ist ein voll ausgestattetes Telekommunikationsprogramm, ideal geeignet für den Zugriff auf elektronische Postdienste, für Vielzweckdienste wie Geonet, und auch für Schwarze-Brett-Dienste - private Informationsdienste von Anwendergruppen und engagierten Computerfreaks. Intalk besitzt eine eigene Kommandosprache namens CCL (Communicatons Control Language). Damit lassen sich Steuersequenzen zur Automatisierung von Kommunikationssitzungen erstell. InTalk unterstützt alle gängigen Filetransferprotokolle, einschließlich XMODEM, Kermit und Crosstalk, und hat auch ein eigenes Übertragungsprotokoll für einfachen Dateiaustausch mit anderen InTalk Anwendern. Ankommender Text kann zuerst in die Windows-Zwischenablage und von dort aus in jede beliebige Windows-Applikation kopiert werden.	Palantir Software Fax: 0045/1/9434780 Softline Company, 7602 Oberkirch	**inTalk**
Zusammen mit INTELLIPHONE, einem Zusatzgerät zur Verbindung von PC und Telefon mit DBP-Zulassung ist das Programm PHONE in der Lage, Puls- und Tonwahl zu realisieren. Die Amtsholung erfolgt durch Erdtastenfunktion oder Ziffernwahl. Auf dem PC wird durch Digitalisierung der Anrufbeantworterfunktion die Sprachausgabe und Sprachaufzeichnung realisiert. Es ist ebenfalls ein Spracheditor integriert, so daß das INTELLIPHONE in Verbindung mit der PHONE Software unter Microsoft Windows auch als Diktiergerät eingesetzt werden kann. Ein Terminsystem mit automatischer Warntonfunktion ist ebenfalls integriert. Eigenschaften: • Kurzwahl über Funktionstasten • Wählen aus eigenen Telefonbüchern • Wählen aus Datenbanken • Notizblockfunktion • Sprachspeicherung- und -ausgabe (Anrufbeantworterfunktion) • Bedienerführung per Maus. Telefonbuch-Funktionen: Kurzwahl laden, Kurzwahl drucken, Datenbank laden (dBase III-Datenbank, IBD-Datenbank, PARADOX-Datenbank, Komma delimited, SDF), Datenbank drucken. Anrufbeantworter-Funktionen: Start, Ende, Parameter, Aufnahme, Wiedergabe, Abhören, Löschen, Status. Notizblockfunktionen: Datei laden, Datei-Export (ASCII), Datei drucken, Suchen, Löschen, Recherche-Kriterien (Name, Erstellungsdatum, Wiedervorlagedatum, Betreff, Stichwort).	IBD GmbH Zieglhüttenweg 33-35 7000 Frankfurt 70	**PHONE**
Das Softwarepaket The Network Courier hat sich einem hardwareunabhängigen Standard verpflichtet. Es arbeitet unter Novell, 3-Com, IBM, Banyan, Western Digital und anderen populären LAN-Systemen - mit oder ohne NetBios. Es besteht die Möglichkeit zwischen einer zeichorientierten- und der grafischen Oberfläche von Microsoft Windows zu wählen. Durch die Unterstützung von »Extended Gateways« können Sie auch Nachrichten mit IBM Profs, DEC All-In-One, VAX-Mail, MCI Mail sowie jedem X400-System austauschen. The Network Courier unterstützt bis zu 150 Anwender pro Server.	Consumers Software, Inc. 314 East Holly Street, Suite 106 USA Bellingham, WA 98225 Datawave S.A,CH - 1258 Perly/Genf Management Software, GB - London SW6 2RZ	**The Network Courier**
Anbindung an IBM Mainframe. VG 3270 erlaubt den Zugriff auf Großrechner-Anwendungen und Datenbanken. Dies wird durch die Emulation des IBM 3270 Color Display Terminals auf einem PC/AT oder PS/2 ermöglicht. VG 3270 läuft unter Microsoft Windows, was dem Anwender ermöglicht zwischen mehreren Anwendungen auf dem PC hin- und herzuschalten sowie mehrere (Sessions) Abläufe in der Mainframe-Umgebung starten zu können. VG 3270 nutzt die Hardware Resourcen des VG Communication and Device Prozessors (CDP). Produktmerkmale: • Emulation des IBM 3278 Mono- und IBM 3279 Color-Displays • Ermöglicht Zugriff auf Mainframe-Anwendungen und -Datenbanken • Anschluß an die VG 9500 oder IBM 5088 Steuereinheit über Modem oder direkt • Telekommunikation von 9600 Baud bis 2 Megabits pro Sekunde • Bildschirmformate der Modelle 2 bis 5 • Vom Anwender konfigurierbare Tastatur und Makroschlüssel • Abspeichern und Aufrufen von Bildschirminhalten im Hauptspeicher und von Disk. • Vom Anwender konfigurierbare On-Line-Hilfe • Mit VG Netzwerk Service Lizenz erhältlicht.	VG Systems, GmbH Bonner Str. 178 5000 Köln 51	**VG 3270**
VG Fast File Exchange ist ein Produkt zur schnellen Datenübertragung zwischen IBM Mainframes die mit dem Betriebssystem MVS oder VM arbeiten und damit verknüpften PCs. FFx ist absolut einfach zu gebrauchen: Eine Liste von Zeichnungs-Dateien wird angezeigt und der Anwender kann ganz einfach die zu übertragenden Dateien auswählen. Es sind lediglich drei Schritte auszuführen, um ein Zeichnungsmodell zu senden oder zu empfangen. Die Dateien können einzeln oder in Gruppen, in Echtzeit oder in Stapeldateien zu einem festgelegten Zeitpunkt (z.B. über Nacht) übertragen werden. Ein Übertragungsprotokoll zeichnet sämtliche Datenübertragungen auf. Produktmerkmale: • Bietet Hochgeschwindigkeits-Datenübertragung zwischen IBM Mainframes und Personalcomputern. • Netzwerklizenz im Angebot • Appliktionsspezifische Versionen für CADAM/Micro CADAM erhältlich.	VG Systems, GmbH Bonner Str. 178 5000 Köln 51	**VG FFx - Fast File Exchange**
VG High Function Graphics ist ein Soft- und Hardware-Produkt. In einem PC/AT, PS/2 oder Kompatiblen installiert, ermöglicht es das Einloggen in Host-Anwendungen, welche auf dem Grafiksystem IBM 5080 basieren. Genauso wie die IBM 5080 High Function Graphics Station, welche emuliert wird, ist auch der Personalcomputer durch eine VG 9500 oder eine IBM 5088 Steuereinheit mit dem Host verbunden. Wenn Sie VG HGF aufrufen, erscheint sofort das High Function Graphics Fenster. Gleichzeitig stellt das Programm die Verbindung zum Host her. Indem Sie das VG 3270 Programm auf Ihrem PC aufrufen, können Sie nun Ihre Grafik-Session direkt auf dem Mainframe starten. Wenn Sie sich nun in den Host eingeloggt haben stehen Ihnen sämtliche Ressourcen und die gesamte Leistungsfähigkeit der Host-Applikationen auf Ihrem PC zur Verfügung. Jetzt können Sie Modelle erstellen, modifizieren, vervollständigen und verwalten - und dies alles direkt von Ihrem Personalcomputer oder Personal System/2 aus. Produktmerkmale: • Erlaubt das Einloggen in den Mainframe um dort Grafik-Sessions vom PC aus zu starten oder zu kontrollieren • Netzwerklizenz verfügbar • Screen Print • Anwenderspezifische Anpassungen.	VG Systems, GmbH Bonner Str. 178 5000 Köln 51	**VG High-Function Graphics**

Produktname	Beschreibung	Hersteller/Vertrieb
	Musik	
M/pc	Interaktives Kompositionsprogramm. M/pc ist nicht nur Voyetras erstes Programm mit der grafisch orientierten Windows-Benutzeroberfläche, sondern auch eine neue Generation von Musik-Programmen. Hier werden auf »spielerische« Art Variationen des eingegebenen musikalischen Materials erzeugt. M/pc ist in jeder Hinsicht ein »Intelligent Musical Instrument«: es fällt automatisch Entscheidungen in Erwiderung auf Realtime-Befehlseingaben. Eine faszinierende Möglichkeit, aus eingespielten Grenzen auszubrechen und Neues zu kreieren. M/pc kann sowohl Song-Dateien aus Sequencer Plus übernehmen, als auch fertige Dateien zur Weiterbearbeitung und Integration in Sequencer Plus-Songs bereitstellen. M/pc ist übrigens die lizensierte Weiterentwicklung der Macintosh-Version »M« von Intelligent Music. Geeignet für alle MIDI-Instrumente. Spezielle Version für YAMAHA-C1 lieferbar.	Voyetra Technologies, Inc. M3C Systemtechnik GmbH, 1000 Berlin
	Zeitplanung	
TIMEGRAF	Leistungsmerkmale: • bis zu 200 Aktivitäten je Terminplan, bis zu 6 Datenketten je Aktivität • Soll-Anfang bzw. Soll-Ende je Aktivität über Formeln zu verknüpfen • Ist-Anfang bzw. Ist-Ende-Markierung • Übernahme von Fremdformaten (ASCII-Dateien, Macintosh-Dateien) • Zeitausschnitt in Tagen, Wochen, Monaten, Quartalen und Jahren • freie Grafik (Text in versch. Schriftarten, Pfeile, Rechtecke, Kreise) • Benutzeroberfläche Windows 2.0 • Ausgabe auf Nadeldrucker, Laserdrucker und Plotter	Computerservice Decker Meisenweg 29 8520 Erlangen netnice & partner, 8555 Adelsdorf
WinTime	Termin-Management per Computer. WinTime ein Terminkalender, der automatisch Terminvorschläge macht und die Zeit überwacht. Ideal im Einsatz für Praxen, Kanzleien, Freiberufler und auf dem Sekretärinnenschreibtisch. Einzelne Termine und Mitarbeiter, Projektdeadlines und Memoranden sind farbcodiert und somit schnell zu unterscheiden. WinTime in Stichworten: • Darstellung jeweils einer ganzen Woche • Zeitintervalle einstellbar von 10, 15, 20, 30, 60 Minuten • Terminkalender bis zum Jahr 2000 +• Einzelpersonen und Gruppen gleichzeitig zu managen • Kapazität: 65.000 Mitarbeiter und Gesprächspartner • Stichworte zu jedem Termin zu vermerken • Ausgabe von Terminlisten, Deadlines für Projekte, etc. • Farbkodierung für verschiedene Termine (Gespräche, Essen, Treffen außer Haus, Urlaub, usw. frei definierbar) • Suchen nach Stichworten • Adress- und Telefonverzeichnis eingebaut • als Netzwerkversion »NetTime« erhältlich	Palantir Software 12777 Jones Road, Suite 100 USA Houston, TX 77070 Tel.: 001/713/955-8880 Softline Company, 7602 Oberkirch
	Programmierung	
Actor	Es gibt eine neue Technologie, die das Entwickeln in der Windows-Umgebung einfacher und leistungsfähiger gestaltet: Objektorientiertes Programmieren. Diese Technologie macht Actor, von der Whitewater Group, zum produktiven und vernünftigen Einstieg in die Windows-Programmierung. Durch das wiederverwendbare Objekt-Toolkit, welches Dialogboxen, Fenstergestaltung und Grafikelemente enthält, wird Ihre gesamte Produktivität mehr als verdoppelt. Mit dem Toolkit haben Sie die Möglichkeit, sofort auf die Fenstergestaltung sowie auf die Verwaltung von grafischen Material und Daten zuzugreifen, um somit schnellste, maßgeschneiderte Anwendungen zu erstellen. Actor stellt eine interaktive, windowsorientierte Entwicklungsumgebung zur Verfügung, die sofortige Kompilierung, interaktives Testen und Quellcode-Debuggen ermöglicht. Es ist eine mit allen Eigenschaften versehene leicht erlernbare Programmiersprache, welche das Erstellen von schnellen Standalone Anwendungen ermöglicht, die den Dynamischen Datenaustausch, Expanded Memory sowie das dynamische Linken zu Microsoft C-Code unterstützen.	The Whitewater Group 906 University Place USA Evanston, IL 60201 Intellis
EasyCASE	Das Programmentwicklungswerkzeug EasyCASE besteht aus den drei Modulen EasyCASE(SD), EasyCASE(DD) und EasyCASE(SP). EasyCASE(SD) unterstützt das grafische Editieren von Kommunikationsplänen. Informationen über Objekte und ihre Beziehungen zueinander werden in einem Data Dictionary verfügbar gehalten. Mit einfachen maus- und menügesteuerten Operationen unterstützt EasyCASE(SD) sehr effizient das Konstruieren und Ändern von Kommunikationsplänen und erspart das mühsame Neuzeichnen nach jeder Änderung. EasyCASE(DD) ermöglicht Standardauswertungen des Data Dictionary. Außer einem hierarchisch und alphabetisch sortierten Inhaltsverzeichnis extrahiert EasyCASE(DD) zu jeder Funktion und jeder Variablen des Systems alle darüber gespeicherten Fakten (z.B. alle Kommunikationspfade) und erstellt eine übersichtliche Liste. Als Data Dictionary dient eine relationale Datenbank im dBase-III-Format, die bei Bedarf auch mit eigenen dBase-Programmen ausgewertet werden kann.	
ENVISION	CASE, Software- und System-Entwicklung, visuelle Datenbank. Envision ist eine schnelle, freundliche und flexible Umgebung zur Entwicklung, Verbesserung und Pflege jeglicher systemorientierter Projekte. Durch den Gebrauch einer State-Of-The-Art Technologie kann der Anwender ENVISION so modifizieren, daß es ihn bei der Entwicklung und Dokumentation von Systemen unterstützt. Als grafische Datenbank stellt es ein schnelles und verlässiges Mittel zum Speichern und Abrufen großer Datenmengen dar. Eine SQL-Schnittstelle ist für den Zugriff auf die Project-Datenbank verfügbar.	Future Tech Systems, Inc. 824 East Main Street Auburn, Washington 98002 USA
HP NewWave Developer Kit	HP-NewWave von Hewlett-Packard ist eine fortschrittliche Anwendungsumgebung, die über eine einheitliche Benutzeroberfläche den Zugriff auf ein komplettes unternehmensweites Informations- und Rechnernetzwerk bietet. Die Anwendungsumgebung ist so gestaltet, daß der Benutzer sich voll auf seine eigentlichen beruflichen Aufgaben konzentrieren kann, anstatt einzelne Anwendungsprogramme erlernen zu müssen. HP-NewWave ist eine offene Umgebung und bietet Software-Entwicklern eine optimale Möglichkeit, ihren Kunden benutzerfreundliche Lösungen zu liefern. Der HP-NewWave Developer-Kit besteht aus drei Komponenten: • NewWave Environment Software (Object Management Facility, Hot Link, Agents, Systemdienst, Encapsulation Tools). • Entwicklungswerkzeugen (Software Bibiliotheken für das Application Programming Interface, Source Code Beispiele). • Dokumentation (Programmer Orientation Guide, Programmer Reference Manual, User Interface Guidelines, Writer`s Style Guide	Hewlett-Packard GmbH Postfach 16 41 6380 Bad Homburg

Beschreibung	Hersteller/Vertrieb	Produktname
Modern Art ermöglicht es einem Anwender, ohne Programmierkenntnisse aufwendige Anwendungen auf dem PC zu entwickeln. Die Programmentwicklung erfolgt grafisch durch die Bedienung mit der Maus. Modulare Entwicklung durch Neugruppierung und Zusammenfassung. Kein Unterschied zwischen Programmentwicklung und -ablauf. Kompilierung von gezeichneten Programmen zu selbständigen optimierten Anwendungen. Die Entwicklungsumgebung basiert auf einer relationalen Datenbank. Modern Art ist die Weiterentwicklung der objektorientierten Programmierung. Die Entwicklungsumgebung stellt eine Reihe von Werkzeugen zur Verfügung, die der Entwickler einfach mit der Maus anklickt. Der Mauscursor ändert sich und stellt das Werkzeug dar. Dies kann neben dem normalen Pfeilcursor ein Greifer, ein Kopierer oder ein Textcursor sein. Es gibt Schere, Staubsauger, Fragezeichen und noch einige andere, deren Bezeichnung und Aussehen auf die Funktion unmittelbar hinweisen. Des weiteren gibt es ein Lager (Parts) mit vorfabrizierten Elementen. Eine Anwendung entsteht, indem man Grundelemente aus dem Lager nimmt und auf die Zeichenfläche positioniert. Ein Schema »läuft« immer, es gibt keine separaten Design- und Entwicklungsmodi. Eine kompilierte Anwendung kann jedoch auch ohne Entwicklungsumgebung ablaufen.	BLITZ Datentechnik GmbH Hainstr. 11 8600 Bamberg Tel.: 0951/200240	**Modern Art**
Das Whitewater Resource Toolkit unterstützt Sie durch einen bequemen und konsistenten Weg, das Aussehen und Ausstrahlung einer Applikation zu beeinflussen, indem es Ihnen hilft, Dialogboxen, Farb-Bitmaps, Cursor und Icons zu erstellen. Es ist ein vollständiges und unentbehrliches Tool für jeden Actor- oder C-Programmierer sowie Systemintegratoren. Zeichnen Sie Dialogboxen, Farb Bitmaps, Cursor und Icons einfach durch Auswählen aus einer reichhaltigen Palette. Erstellen Sie Menüs, Tastaturbeschleuniger und String-Resourcen. Sie können fremdsprachige Versionen mit den String- und Tabellen-Editoren erstellen ohne, den Quellcode der Anwendung haben zu müssen. Das Whitewater Resource Toolkit enthält nicht das Microsoft Windows Software Development Kit oder den Resource Compiler. Das Produkt ist kompatibel mit allen Windows Resource-Dateiformaten und kann RC kompatible Script- und Bitmapdateien generieren. Das Whitewater Resource Toolkit ist in Actor programmiert.	The Whitewater Group 906 University Place USA Evanston, IL 60201 Intellis	**Whitewater Resource Toolkit**
WinTrieve, entwickelt von der Whitewater Group, ist ein Programmierer-Tool. Es bietet Ihnen ein vollständiges Datei-Management für jedes Windows-Programm das Sie entwickeln. WinTrieve verwaltet große Datenmengen besonders effektiv, indem es diese in ISAM-Dateien abspeichert. WinTrieve eignet sich hervorragend bei der Datenverwaltung bei Applikationen wie Inventur-, Fertigungs-, Auftragseingangs- und Fakturierungsanwendungen, wie auch bei technischen und wissenschaftlichen Anwendungen. Mit WinTrieve können Sie Datensätze definieren, die mit einem Index auf jede Zelle versehen werden. Es enthält außerdem erweiterte Funktionen die Ihre Daten schützen. Ferner können Sie mehrere Datei-Operationen in einer zusammenfassen. Sie können ebenfalls alle Änderungen, die Sie in Ihrer Datenbank vornehmen, festhalten, indem Sie sich der automatischen Aufzeichnungsmöglichkeiten von WinTrieve bedienen. WinTrieve bietet Schnittstellen zu Microsoft C oder dem Whitewater Group Actor. WinTrieve enthält ein vollständiges Lernprogramm und eine Programmers Reference über das Index-Dateimanagement und die Bibliotheken der C- und Actor-Funktionen.	The Whitewater Group 906 University Place USA Evanston, IL 60201 Intellis	**WinTrieve**

Textverarbeitung und -retrieval

Beschreibung	Hersteller/Vertrieb	Produktname
Ami gehört zu den Textverarbeitungsprogrammen, die deutlich an DTP-Programme angelehnt sind. Texte werden in Style-Sheets geschrieben. Die Bearbeitung von Texten ist im Draft-Modus und in verschiedenen Grafik-Modi möglich. Da Ami eng mit den geladenen Druckertreibern zusammenarbeitet, zeigt das Programm in allen Grafik-Modi den Text so an, wie er auf dem Papier erscheint. Da das Programm dem SAA-Standard folgt, dürfte den Anwendern die Einarbeitung sehr leicht fallen.	Samna	**Ami**
Dragnet ist ein Text-Retrieval-Programm für Microsoft Windows und im Hintergrund lauffähig. DragNET sucht in den Dateien auf der Festplatte gezielt nach Stichworten und zeigt die Fundstellen sekundenschnell mit Namen auf dem Bildschirm an. Es genügt das kleinste Stichwort oder sogar nur ein Teil dessen; die Arbeit wird durch »Joker« erleichtert. Ein Maier läßt sich z.B. so finden (M***r). Ergebnis der Suche: Meier, Maier, Mayer, Meyer.	ACCESS SOFTEK 3204 Adeline Berkeley, CA 94703 USA	**DragNET**
Mit HYPARCHIV sind im Büroalltag erhebliche Organisationsverbesserungen zu erzielen. Einfache Dokumentenerfassung (Schriftstücke, Pläne, Bilder) mittels Scanner, schnelles Auffinden von Dokumenten mit stets vollständigen Vorgängen und ein Höchstmaß an Datenschutz. Außerdem besteht die Möglichkeit der Weiterverarbeitung im Rechner. Dieses System ist ein wesentlicher Fortschritt auf dem Weg zum »papierarmen Büro« der Zukunft. Die Benutzerführung ist zwischen Deutsch, Englisch und Französisch umschaltbar. Das äußerst flexible, netzwerkfähige HYPARCHIV-System unterstützt sowohl sehr große Magnetplatten, wiederbeschreibbare magneto-optische Speicher als auch verschiedene Einzellaufwerke und Jukeboxes in WORM-Technologie. Die ACS Systemberatung bietet sowohl komplette Systeme als auch die Entwicklung kundenspezifischer Lösungen an.	ACS GmbH 2000 Hamburg	**HYPARCHIV**

Utilities

Beschreibung	Hersteller/Vertrieb	Produktname
Die Altos-Benutzeroberfläche APEX-(Application Executive) ermöglicht die simultane Benutzung aller im AdLANtes-Netz verfügbaren Applikationen unter UNIX, MS-DOS oder Großrechner-Betriebssystemen. Die Benutzerführung erfolgt mit Fenstertechnik (Windows, Icons) und Maus. Mit APEX können nicht nur mehrere Applikationen gleichzeitig an einem Arbeitsplatz genutzt, sondern auch Daten zwischen ihnen ausgetauscht werden. UNIX und MS-DOS-Programme können sogar auf gemeinsame Datenbestände zugreifen, da UNIX- und MS/DOS-Dateien im gleichen Dateisystem (keine Partitions) verwaltet werden. Datenübernahme mit SNA oder Datex-P erfolgt per File-Transfer oder interaktiv im »Cut & Paste«-Verfahren von Fenster zu Fenster. Die APEX-Oberfläche ist individuell pro Benutzer änderbar. Jeder Benutzer findet nach der Anmeldung am System seinen gewohnten Satz an Icons und Menüs vor. Features: • Bis zu 7 Fenster mit UNIX- oder Großrechner-Applikationen an einem Arbeitsplatz • MS/DOS-Fenster nur durch lokal verfügbaren Hauptspeicher begrenzt • Maus-Schnittstelle • Gemeinsame Dateien für MS/DOS und UNIX • Datenaustausch zwischen den unterschiedlichsten Applikationen	ALTOS COMPUTER SYSTEMS Würmstr. 55 8032 Gräfelfing Tel.: 089/85484-0 Altos Computersystems, 8032 Gräfelfing Altos Computersystems, 2000 Hamburg	**APEX**

Produktname	Beschreibung	Hersteller/Vertrieb
artus	Die Hauptanwendung von artus ist die Nachbearbeitung von gescannten Bildern. Es kann aber auch als leistungsstarkes Zeichenprogramm mit außergewöhnlichen Fähigkeiten eingesetzt werden. Hier sollen nur kurz einige der besonderen Funktionen von artus aufgeführt werden: • echte 256 Grauwerte pro Bildpunkt frei wählbar (Farbe in Vorbereitung) • nachträgliches Ändern von Kontrast und Helligkeit ohne Bildinformationsverlust • Kanten extrahieren (Bild schärfer machen) und eliminieren (Bild verschwimmen lassen) • Weichzeichnen und diverse andere Effekte • Zoomen bis zur 16-fachen Originalgröße bzw. beliebige Verkleinerung • alle Bearbeitungsfunktionen stehen in jeder Zoomstufe zur Verfügung • Stempelfunktion mit selbstdefinierten Stempeln • Bearbeitung beliebiger, nicht rechteckiger Bildausschnitte • Export der Bilder in den gängigen Formaten (TIFF, Postscript, GEM-Image) • Verarbeitung von Bilddateien von bis zu 16 Mbyte (entspricht DIN A4 bei 400 dpi) • ständige Darstellung eines Übersichtsbildes • Benutzerführung deutsch mit Hilfsfunktion (auch andere Sprachen optional).	
Bridge/386 und Bridge/OS	Diese Software ermöglicht die Integration von verschiedenen Softwareprodukten unter der gemeinsamen Benutzerschnittstelle von Windows. Sie gibt dem PC-Verwalter und auch dem Benutzer die Möglichkeit, eigene Software zur Steuerung von Anwendungen zu entwickeln. Es arbeitet unter jedem NetBIOS-kompatiblen LAN, wodurch es Benutzeranwendungen transparent in einem Neutzwerk vereinigen kann, während der Benutzer eine scheinbar lokale Arbeitsumgebung beibehält. PC-Verwalter können so dem Anwender eine Menge der Komplexität von Netzwerken abnehmen. Bridge/386 ist eine speziell für Windows entwickelte Batchsprache, die es dem Benutzer erlaubt, wiederholende Windows-Abläufe zu automatisieren und sich mehrerer Anwendungsprogramme zu bedienen, ohne Windows zu verlassen. Es eröffnet Möglichkeiten, vergleichbar denen mit der DOS-Batchsprache und erweitert Windows um zahlreiche Möglichkeiten. Mehr Details über Bridge im Microsoft System Journal Nov./Dez. 1988	
Btx-Fenestra	Btx-Fenestra ist ein Zusatzsoftware-Paket für den Rafi Hardware-Decoder (PC-Btx Adapter). Btx-Fenestra ermöglicht den Btx-Betrieb unter MS-Windows mit den bewährten Vorzügen wie Multitasking-Betrieb und Datenaustausch mit anderen Windows-Applikationen (z.B. Übergabe von Börsendaten an Excel, Übernahme von Texten aus Excel, Write, etc. und umgekehrt). Der EMS-Speicher wird von Btx-Fenestra vollständig angesprochen. Die Abläufe in der Btx-Umgebung werden, wie unter Microsoft Windows gewohnt, durch leicht zu bedienende Pull-Down-Menüs gesteuert. Die wichtigsten Funktionen: • Speichern und Drucken von Btx-Seiten als Grafik oder Text, Seitenverwaltung mit Löschen, Kopieren und Umbenennen, Diashow, Laden von Telesoftware • Erstellen von Kurzwahllisten für häufig aufgerufene Seiten oder Einstiegsprozeduren in Externe Rechner • Erstellung von automatischen Abläufen mit Hilfe von deutschen Makrobefehlen, Verknüpfung von Makros für komplexe Abläufe • Standard-Makros werden mitgeliefert, z.B. für automatische Anwahl, Briefkasten leeren, Telexversand, Börsenkurse auslesen, Homebanking.	
Btx-Manager V.3.0	Unterstützung des Btx-Dialogs: Btx-Grundfunktionen wie Anwahl, Abwahl, Seiten speichern im Text- oder Grafik-Format, Seite drucken im Text- oder Grafik-Format, Seite löschen, kopieren, umbenennen; bietet Anzeige der Online-Zeit, Erstellung von Kurzwahl-Listen, Suchfunktion und vieles mehr. Automatisierung des Btx-Dialogs: • Makro-Sprache für alle Btx-Funktionen • deutsche Befehle • Befehlsfolge wie in der manuellen Bedienung • automatische Generierung von Makros • Laden von Standard-Makros beim Start der Software • Mitlieferung von Standard-Makros wie Automatische Anwahl, Briefkasten leeren, Mitteilung versenden, Btx-Telex versenden/empfangen, Regionalbereichswechsel • Angabe eines Makros, das bei Programmstart ausgeführt werden soll • Verknüpfung von Makros für komplexere Abläufe (z.B. Hombanking, Börsendaten lesen, etc.) • Zusammenspiel mit anderen Windows-Applikationen (z.B. Übergabe von Börsendaten an Excel, Übergabe von Grafiken in Page-Maker oder MS-Paint, Übernahme von Texten aus Write und Einspielen als Btx-Mitteilung.	
clip&connect	Clip&connect - die neue praktische Erweiterung von MS-Windows. Sie kopieren Texte, Grafiken, Tabellen, gescannte Bilder - aus vielen verschiedenen Dateien um sie bequem vom clip&connect-»Vorratslager« abzurufen und in neue Dateien einzufügen. Im Netzwerk wird clip&connect zur elektronischen Post. Sie klicken »EXPORT«, senden Ihr Rundschreiben an verschiedene Abteilungen und haben somit alle Mitarbeiter gleichzeitig informiert. Ein grafisches Symbol auf deren Bildschirm signalisiert die Ankunft des elektronischen Rundschreibens. Special Features: • Add-on für alle MS-WINDOWS-Anwender • erweitertes Windows-Clipboard • bis zu 20 »Clips« • Benutzeroberfläche: ausschließlich MS-Windows Standardfunktionen • 100 % kompatibel zu allen existierenden und zukünftigen MS-Windows-Applikationen • Multitaskingfähig unter MS-Windows/386 • für Standalone und Netzwerk-Betrieb geeignet • Clipboard SAVE (automatischer Zwischenspeicher für Clipboard Daten zum Anlegen einer »clipboard library«) • Desktop COMMUNICATION (freier Daten- und Informationsaustausch im Netzwerkbetrieb) • ELECTRONIC MAIL (einfach zu nutzendes E-Mail-System zum Nachrichtenaustausch im Netzwerkbetrieb).	SPI
ColorLAB	Einlesen von Realbildern mit den Sharp Farbscannern JX-450 und JX-300. Einsetzbar für Desktop Publishing, Präsentationsgrafik, Bilddatenbanken, Screenshows, Werbung, Marketing und Präsentationen allgemein. ColorLAB ist die erste Farbscannersoftware für Windows. Die Prescan-Funktion ermöglicht schnelles Arbeiten. 256 Farben aus 16,8 Millionen sind mit der Auflösung von 75 bis 300 Punkten pro Zoll darstellbar. Die Auflösung läßt sich in Höhe und Breite variieren. Die gescannten Bilder können in Helligkeit, Kontrast, Farbwert und Schärfe korrigiert werden.	
ComfoBridge	Vieles wird einfacher, und überhaupt erst möglich. Batch Programmierung. Brücke zu DOS, individuell gestaltbare Menüs. ComfoBridge ist ein völlig neues Programmiertool für grafische Benutzeroberflächen, das erstmals eine komfortable Verbindung von DOS- und WINDOWS-Applikationen ermöglicht. Erstmals können nun alle klassischen DOS-Anwendungen einwandfrei unter der Windows-Benutzeroberfläche eingesetzt werden. Erstmals ist Batch-Programmierung unter Windows möglich. Erstmals kann man einfach und schnell Windows-Menüs individuell gestalten. Erstmals können beide Welten in Netzwerksystemen verbunden werden.	SPI
Jaws! - Hammer & Wise Tools	Jaws ist ein Datenkompressor, der den Speicherbedarf von Dateien reduziert, die unter Windows abgelegt werden, um bis zu 50%. Ideal für Desktop Publishing-Anwender, deren gescannte Dateien sonst riesige Plattenkapazitäten blockieren würden. Der Hilfsmenü-Generator leistet Programmierern und PC-Support-Managern in Großfirmen wertvolle Dienste beim Umgang und der Schulung von Windows-Programmen. Mit geringem Aufwand können individuelle Hilfsfenster für jedes Programm erstellt werden. Ein einziger Befehl ruft die Hilfsanzeige auf, deren endloser Inhalt beliebig durchgeblättert werden kann.	R Company 2170 Georgina Ave. USA Santa Monica, CA 90402 Softline Company, 7602 Oberkirch
Opus 2.0 Hypergraphics	Software vergleichbar mit HyperCard auf dem Macintosh. Grafiken und Text werden über »elektronische Verbindungen« Hyperlinks verknüpft.	RSI Roykore Software, Inc. Softline Company, 7602 Oberkirch Siener Soft, 6200 Wiesbaden Digital Image Systems Intellis, CH - Ebmatingen

Beschreibung	Hersteller/Vertrieb	Produktname
Elektronischer Notizblock und Kalender für Windows. Automatisch überwacht PIM alle Termine, listet Stichworte und Notizen zum jeweiligen Zeitpunkt auf und druckt saubere Terminplaner.	Softline Company, 7602 Oberkirch	**PIM Personal Information Manager**
Pizazz macht einen »Schnappschuß des Bildschirms«: Wenn die PrtSc-Taste gedrückt wird, unterbricht Pizazz den Ablauf des Programms. Daraufhin können im Hauptmenü folgende Funktionen ausgewählt werden: • Bildschirm (auch farbig) auf Nadel- oder Laserdrucker ausdrucken (mit bis zu 200 Farb- oder Graustufen). • Bildschirm auf Diskette abspeichern oder an MS-Word und Desktop Publishing-Programme exportieren (als TIFF- oder PCX-Datei). Grafik, Text oder beides gemeinsam lassen sich mit Pizazz verarbeiten. Egal, ob ein Lotus-Diagramm, eine Bildschirmabbildung für ein Handbuch, ein Grundriß aus einem CAD-Programm oder sonst etwas schnell festgehalten werden soll. Als weitere Optionen stehen zur Verfügung: Auswahl des Bildschirmausschnitts, Vergrößern von Ausschnitten bis max. DIN A4-Größe, gedreht drucken (90-Grad Rotation), Smoothing-Funktion (Weichzeichner) zum Glätten von »Treppchen« bei Diagonalen und Buchstaben, sowie Farbumkodierung bei Nadeldruckern (z.B. NEC P5 Color).	Softline Company, 7602 Oberkirch	**Pizazz Plus**
Prompt! ist ein Festplatten-Datei-Manager für Microsoft Windows-Anwender. Es bietet Ihnen eine grafische Darstellung des Verzeichnis-Baumes. Sie können Verzeichnisse erstellen oder umstellen, indem Sie direkt den Verzeichnis-Baum verändern. Prompt! stellt Ihnen auch eine Suchfunktion zur Verfügung, mit der Sie nach verlorenen Dateien suchen können. Sie können nach dem Dateinamen, der Erweiterung (Suffix) oder nach dem Datum der Erstellung suchen. Sie können Anwendungen aufrufen und Dateien entweder aus der Such-Ergebnis-Box oder aus dem Verzeichnis-Baum laden. Sie können jede Datei oder Dateigruppe kopieren, umbenenen, verschieben oder löschen, indem Sie sich der Prompt!-Dateibearbeitungsbefehle bedienen. Diese Befehle können Sie auf Dateien im Verzeichnis-Baum sowie auf Dateien in der Such-Ergebnis-Box anwenden. Prompt! beinhaltet auch eine View-Option, Sie können sich erst den Inhalt der Dateien ansehen, bevor Sie sich entscheiden, die Datei weiter zu bearbeiten.	ACCESS SOFTEK 3204 Adeline Berkeley, CA 94703 USA	**Prompt!**
Für Typografen, Werbeagenturen, Drucker, Grafikdesigner. Mit Publisher's Type Foundry können Sie Zeichen nach Ihren Wünschen modifizieren, Symbole entwerfen oder ganze Fontbibliotheken von einer einzigen Typenform erstellen. Es besteht die Möglichkeit, verschiedenste Arten von Fonts in Publisher's Type Foundry einzulesen um diese zu modifizieren: Windows Screenfonts, Standard Bitmap Fonts von DTP- oder Textverarbeitungsprogrammen, Downloadfonts für Laserdrucker oder Postscript- und kompatible Fonts. Das Programm enthält zwei Editoren. Den Outline Editor, dieser läßt Sie den Umriß eines Zeichens oder Symbols festlegen, sowie den Bitmap Editor, mit dem Sie jedes Zeichen für optimale Druckausgabe feinbearbeiten können - Bildpunkt für Bildpunkt. Publisher's Type Foundry enthält ein weiteres innovatives ZSoft-Produkt, PC Paintbrush für Windows, ein Mal- und Zeichenprogramm.	ZSoft Corp. 450 Franklin Road/Suite 100 USA Marietta, GA 30067 Softline Company GEPO-Soft	**Publisher's Type Foundry**
Geben Sie Ihrem IBM PC/AT; PS/2 oder Kompatiblen ein neues Aussehen - mit dem PubTech File Organizer. Jetzt können Sie die Leistungsfähigkeit Ihres Computers mit der intuitiven, auf Icons basierenden Arbeitsoberfläche nutzen. Sie haben die Computer Funktionen wie auf einem Schreibtisch vor sich ausgebreitet - leicht zu verstehen und fertig für Sie um diese nun allein mit der Maus zu steuern. Starten Sie häufig benötigte Abläufe direkt durch einen festgelegten Tastendruck. Erstellen Sie Ihre eigenen »Hot-Key« Menüs. Verwalten und steuern Sie Ihre Laufwerke und Drucker. Führen Sie Backup- und Restorebefehle von ausgewählten Dateien oder ganzen Verzeichnissen aus. Lernen Sie schneller, indem Sie Ihre Tastatur umbelegen, um Systeme zun emulieren, mit denen Sie bereits vertraut sind. Zwischen den Icons und Pull-Down-Menüs haben Sie alle Funktionen übersichtlich präsentiert - für sofortigen und leicht verständlichen Gebrauch.	Publishing Technologies, Inc. 7719 Wood Hollow Drive, Suite 260 USA Austin, TX 78731 CfM Computertechnik AG	**PubTech File Organizer**
Screen Grab macht Bildschirmschnappschüsse von praktisch allen Windows-Programmen. Alle Grafikkarten und Bildschirme, die MS Windows nutzt, werden unterstützt - inklusive Ganzseitenschirme und hochauflösende Grafikkarten (z.B. 560 x 800 Pixel). Die Schnappschüsse können auf Nadel- und Laserdrucker (HP oder Postscript) ausgedruckt oder als TIFF- , IMG und PCX-Dateien gespeichert werden.	Softline Company, 7602 Oberkirch	**Screen Grab**
Der VG Data Manager VG DMR ermöglicht die Verwaltung von Micro CADAM Zeichnungs-Dateien durch die Maus. Dies befreit den Anwender vom ständigen sich Erinnern an Dateinamen und das Tippen von solchen. Besonders hilfreich ist VG DMR in einer Zeichnungs-Dateiverwaltung innerhalb eines LAN. VG DMR stellt Ihnen sechs Grund-Dateiverwaltungs-Funktionen zur Verfügung: • Anzeige von Gruppen und Anwendernamen • Anzeige, Kopieren, Verschieben, Sortieren und Attributänderungen von Zeichnungsdateien. Zusätzlich gibt es noch verschiedene Features die das Finden und Übertragen von Zeichnungs-Dateien erleichtern. 1. Die Grundfunktionen arbeiten auch in den verschiedensten Kombinationen von Gruppen und Anwendern, sie sind nicht an eine bestimmte Gruppe oder Anwender gebunden. 2. Die Zeichen * und ? können bei der Auswahl von Gruppen, Anwendern, Zeichnungs-Namen, Daten, Kommentaren sowie Plot-Dateien verwendet werden. 3. Eine Folge von Anwenderaktionen kann gespeichert und für wiederholte Ausführungen abgerufen werden. Zusätzlich wird ein Übertragungsprotokoll augezeichnet, das die Kontrolle und eine spätere Prüfung ermöglicht.	VG Systems, GmbH Bonner Str. 178 5000 Köln 51	**VG Data Manager**
Das Bild-Archiv-System - Digitale Videobildverarbeitung für Bilderfassung, Bildbearbeitung, Bildarchivierung, Bildausdruck. VideoMaker bedeutete, mit Hilfe einer Videokamera laufende oder stehende Bilder aufzunehmen. Diese Bilder sind auf dem Kontrollmonitor eines Microcomputers sichtbar, werden digitalisiert abgespeichert und können mit Bearbeitungsfunktionen retuschiert und nachbearbeitet werden. Eine Formularfunktion »Publishing« erlaubt das Mischen von Bildern mit bestehenden Stammdaten - z.B. aus einer Artikelstammdatei. Der Ausdruck von Bildern mit Texten erfolgt auf einem angeschlossenen Laserdrucker oder durch die Belichtung über eine Linotronic mit sogenannten »Postscript-Dateien«. Die Einbindung der Bilder in Desktop-Publishing-Programmen wie PageMaker oder Ventura Publisher (ab Version 2.0) eröffnen vielfältige Einsatzmöglichkeiten.	orgaplus Vertriebs GmbH Kolpingstr. 12-16 7100 Heilbronn Tel.: 07131/51034	**Video Maker**
Mit VG Plotbuf haben Sie die Möglichkeit, gleichzeitig zu plotten und weiter an Ihrer CAD Anwendung zu arbeiten. Indem Plot-Dateien in einen unabhängigen Coprozessor mit Puffer gespoolt werden, hält VG Plotbuf den PC für weitere Bearbeitung frei, während gleichzeitig geplottet wird. Der Plot-Queue Puffer kann mehrere Plot-Dateien gleichzeitig verwalten. Durch einen leicht zu bedienenden Queue Manager, der unter Microsoft Windows läuft, können Sie den Plot-Dateien Prioritäten zuorden oder diese wieder aus dem Spoolbetrieb herausnehmen. VG PLOTBUF nutzt den Communication and Device Processor (CDP), um die Plot-Dateien unabhängig vom PC abzuarbeiten. Dadurch, daß nun der gesamte Task zur Plottersteuerung von dem CDP Adapter und nicht vom PC übernommen wird, steigert sich die Anwenderproduktivität erheblich. Die Ersparnisse beim Gebrauch von VG PLOTBUF können beachtlich sein. Produktmerkmale: • Erlaubt dem Anwender am PC weiterzuarbeiten wärend gleichzeitig geplottet wird • Vergeben von Prioritäten und umstellen der Plot-Queue ist äußerst einfach • Verkürzt nicht produktive CPU-Zeit um 95%.	VG Systems, GmbH Bonner Str. 178 5000 Köln 51	**VG PLOTBUF**

Scanner-Software

Produktname	Beschreibung	Hersteller/Vertrieb
RECOGNITA	Software-Paket für optische Zeichenerkennung mit IBM PC, PC/XT, PC/AT und Kompatiblen. RECOGNITA erspart das neuerliche Eintippen jeder Art von maschinengeschriebenem oder gedrucktem Material. Damit ist das lästige Neuschreiben des bereits vorhandenen Materials nicht mehr notwendig, Zeit kann eingespart werden und Fehlerquellen - durch Neuschreiben bedingt - werden eingeengt. RECOGNITA liest auch Kopien. RECOGNITA liest alle Zeichen ein, die man auf einer üblichen Tastatur finden kann und erkennt darüber hinaus auch die jeweiligen nationalen Zeichen. Im Trainingsmodus können auch weitere Zeichen in den Zeichensatz aufgenommen werden. Eine A4-Seite wird, abhängig von der Druckqualität, in 25-35 Sekunden eingelesen (= 120 Seiten pro Stunde). Der Zeilenabstand wird automatisch erkannt. Optionen: Aufnehmen von neuen Alphabeten, Aufnehmen von neuen Schriftarten, Anpassung an andere Scanner, Anpassung an andere Grafik-Karten, Anpassung an neue Textverarbeitungs- und Desktop Publishing-Systeme.	Szki Computer research und Innovation Center Donati u. 35-45 1015 Budapest Tel.: (361) 350180
ScanDo	Hammerlab ScanDo macht Ihnen die Arbeit mit Ihrem Scanner in Verbindung mit dem Aldus Pagemaker und Micrososft Windows so einfach wie noch nie. Mit ScanDo können Sie bis zu 256 Graustufen scannen, bearbeiten und speichern - und das auf jedem Bildschirm. ScanDo verkleinert und zoomt äußerst schnell. Gleichgültig welche Bildansicht Sie gewählt haben, es stehen Ihnen jederzeit alle Bearbeitungswerkzeuge zur Verfügung - selbst wenn Sie das Bild so verkleinert haben, daß es vollständig auf dem Bildschirm abgebildet ist. Unbegrenzte »Bearbeiten Rückgängig«-Stufen geben Ihnen die Möglichkeit einige oder alle Änderungen die Sie an einem Bild vorgenommen haben Rückgängig zu machen. ScanDo ermöglicht es Ihnen auf quadratische Bilder im Verhältnis 1:1 auf dem grafischen Bildschirm anzuzueigen, damit quadratische Bilder auch quadratisch aussehen und nicht rechteckig. ScanDo liest und schreibt im TIFF, PCX, PCC und MSP Dateiformat ohne auf externe Konvertierungsroutinen zurückgreifen zu müssen. Sie müssen nur das gewünschte Dateiformat mit der Maus anklicken - das ist alles. ScanDo unterstützt Hewlett Packard, Canon, Panasonic, Princeton und Ricoh Scanner.	Hammerlab Corporation 938 Chape Street New Haven, CT 06510 USA
WinScan	WinScan steuert Scanner unter MS-Windows und liest gescannte TIFF-Dateien ein, um sie z.B. per Clipboard an den Graphic & Illustration Designer zu übergeben. Im Programm sind wählbar: • Auflösung: 75, 100 150, 180, 200, 240, 300 dpi • Helligkeit, Kontrast und Pixelgröße. WindowsSCAN spricht alle gängigen Scanner unter MS Windows an. So sind alle Betriebsmodi von z.B. Canon, HP ScanJet, Panasonic, Ricoh, Princeton mit Graustufen und Halbtonmodi aktivierbar. WindowsSCAN bietet komplette Editierfunktionen aller Grafiken mit Pinsel oder Paintbrush. Grafikformate: TIFF und TIFF komprimiert, MSP, PCX. Dateiformate: TIFF-Import	Palantir Software 12777 Jones Road, Suite 100 USA Houston, TX 77070 Softline Company, 7602 Oberkirch

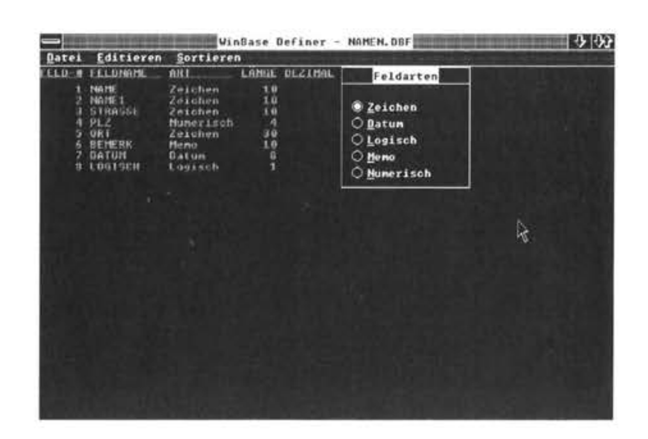
Sonstiges

Produktname	Beschreibung	Hersteller/Vertrieb
Design/IDEF	Design/IDEF ermöglicht die Erstellung von Modellen, die aus Hunderten von Diagrammen bestehen. Automatische Positionierung von Datenflußbeschriftungen einschließlich der ICOM-Beschriftungen auf jedem Diagramm. Automatisch werden IDEF-Aktivitäten numeriert, Pfeilvereinigungen, Pfeilverzweigungen, Brücken und Tunnel erzeugt und Pfeile parallel verlegt. Automatisches justieren von rechten Winkeln in Pfeilen, wenn Objekte verschlossen werden. Jedem Knoten und jeder Verbindung kann Text zugeordnet werden. Bei der Erzeugung von Datenflußpfeilen werden Grammatik und Konsistenz automatisch geprüft. Ein integriertes Data Dictionary erlaubt die Definition von Datensätzen für jede Aktivität und jeden Datenfluß. Automatische Erzeugung von Textdateien, die Diagramm- und Dictionary- Informationen enthalten.	Meta Software Corporation 150 Cambridge Park Drive USA Cambridge, MA 02140 C.I.T. GmbH NOSER AG
HERMES	Angebote und Rechnungen können mit HERMES direkt aus einem Aufmaß heraus erstellt werden. Die Positionsbeschreibungen und die Positionspreise werden entweder manuell erfaßt oder aus einer Stammdatenverwaltung entnommen. Es steht eine Datenbank für den Artikelstamm und eine Datenbank für den Leistungsstamm zur Verfügung. Die Datenbank für den Artikelstamm ist entsprechend der DATANORM aufgebaut. Somit ist es möglich, Katalogdaten von Zulieferern einzuspielen. Es wird ein Leistungsstamm für das jeweilige Gewerk mitgeliefert, das die wichtigsten Leistungstexte enthält. Selbstverständlich können Leistungstexte und Kalkulationen auch selbst erfaßt, verändert oder gelöscht werden. Die Anzahl der speicherbaren Artikel bzw. Leistungstexte ist praktisch unbegrenzt (1,6 Millionen). Für die Anwendung bei Zimmereien ist ein Holzlistenprogramm enthalten. Die Windows-Technik ermöglicht Multitasking, d.h. die Bearbeitung mehrerer Projekte gleichzeitig an einem Bildschirm.	Theobald EDV-Beratung GmbH Aspacher Str. 43 7150 Backnang Tel.: 07919/66158
Telesoft	Reisebüros, Busunternehmen, Theaterkassen, etc. Über Telesoft können hunderte von Veranstaltungen und tausende Vorstellungen vieler voneinander unabhängiger Veranstalter gleichzeitig laufen und somit die Verwaltung von Millionen Einzelplätzen übernommen werden. Bei Bedarf können über den Telefonverkauf von Eintrittskarten hinaus differenzierte Buchungen und Reservierungen aller Art vorgenommen werden, von der Belegung ganzer Passagierflugzeuge bis hin zum Einzelsitz im Strandkorb. Ein umfangreicher Suchwortkatalog ermöglicht schnellsten Zugriff auf jede Information. Maximalleistung: In einem einzigen Vorgang können 256 Plätze gleichzeitig gebucht werden. Bis zu 144 Tickets in der Minute werden sofort in grafisch hochwertiger Qualität von schnellen Desktop-Laser-Druckern hergestellt. Sämtliche Auftragsdaten sowie Reservierungen für Monate und Jahre im voraus werden in der transaktionsgeschützten Datenbank sicher gespeichert. Damit sind irrtümliche Doppel- oder Nullbelegungen ausgeschlossen. Alle Reservierungen werden im Echtzeitdialog zwischen PC und Großrechner vorgenommen. Die Kassen führen automatisch Sofort-Faktura durch.	Teleticket GmbH Weidestr. 118 B 2000 Hamburg 76 Tel.: 040/27075270 c.a.r.u.s. Computer Service GmbH

dBase-Daten unter MS-Windows verwalten:

Schneller Zugriff garantiert

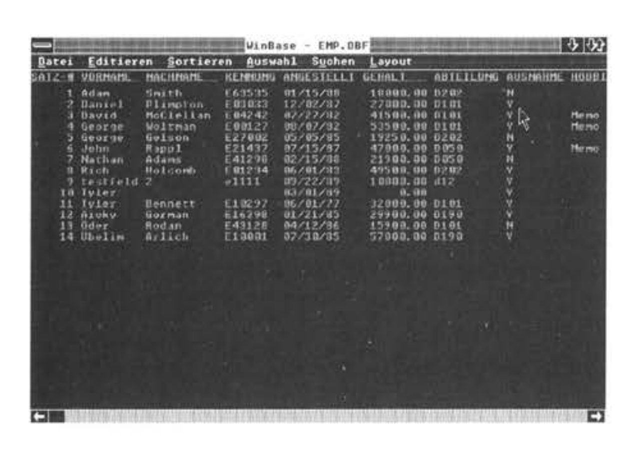
Bisher war es unter MS-Windows nicht möglich, dBase-Datenbanken zu erstellen und zu bearbeiten. Die einzige Möglichkeit, dBase-Daten in eine MS-Windows-Applikation zu bekommen war bisher, die Daten über einen Import zu transferieren. Das soll nun anders werden. Die Firma Pioneer Software entwickelte ein Programm, daß auch dem Ungeübten die Arbeit mit dBase-Datenbanken und Daten unter MS-Windows ermöglicht. Vorteile dabei: Sie brauchen kein dBase und können sich das Erlernen der dBase-Programmiersprache sparen.



◄ *Bild 1: Windef.*

Das Programm Winbase ist für 98 Mark als »Bookware« beim Verlag Markt&Technik zu beziehen und soll laut Handbuch für alle dBase-Versionen (außer dBase IV), Foxbase und Clipper geeignet sein. Ein alter dBase-Kenner würde wohl sagen, daß es sich bei diesem Programm um einen »intelligenten BROWSE-Befehl« handelt. Damit wäre der Leistungsumfang des Produktes einigermaßen genau umschrieben. Unter der MS-Windows-Benutzeroberfläche läßt sich mit Hilfe der Maus und sehr schnell eine Datenbank erstellen, eine Datenbank ändern und auch löschen. Für diese Arbeiten wird das mitgelieferte Programm WINDEF.EXE benutzt, das speziell für diese Aufgaben sowie das physikalische Sortieren und das »Packen« von als gelöscht markierten Daten zuständig ist. Durch dieses Programm ist es möglich, daß gesamte Programmpaket als eigenständiges Datenbanksystem zu benutzen. Natürlich ohne die Programmiermöglichkeiten des kompletten dBase.

Winbase ist, wie unter MS-Windows üblich, mit der Maus zu bedienen, was einiges an Tipparbeit spart. Einige kleine Einschränkungen fallen gleich zu Beginn auf. So können zum Beispiel Memo-Dateien in Clipper weitaus größer als die von Winbase unterstützen 4096 Byte werden. Was mit dem Rest passiert, verschweigt das Handbuch. Zeichenfelder werden von Winbase bis zu einer Länge von 254 Zeichen unterstützt, Clipper dagegen bringt es auf 32 Kbyte. Als typisch amerikanischen Schwachpunkt stellt sich die Eingabe von Datum-Werten dar. Obwohl das Produkt angeblich alle »erdenklichen« Datumformate unterstützen soll, klappt nur die Eingabe im amerikanischen MM/TT/JJ Format. Logische Felder enthalten immer nur die Werte Y/N oder T/F. Zu erwähnen ist noch, daß bei der Sortierung die Umlaute ans Ende der Datei gestellt werden, was nicht den deutschen Sortier-Regeln nach DIN entspricht. Nicht vergessen werden soll, daß Datenbanken unter Winbase nicht indiziert werden können. Für eine schnelle Suche in einer großen Datenbank ist es aber wesentlich, die Daten mit Hilfe eines Index in die gewünschte Reihenfolge zu bringen, um kurze Suchzeiten zu erreichen. Unter Winbase bietet sich nur die Möglichkeit der Sortierung an, was bei großen Datenbanken viel Zeit benötigt.



▸ *Bild 2: Datenbank.*



▸▸ *Bild 3: Bedienungsmenü.*

Winbase in der angebotenen Form stellt sich als die Version 1.35 dar. Für mehr Leistung wie zum Beispiel dynamischer Datenaustausch (DDE) mit Excel und der SQL-Programmiersprache sind erst ab der Version 2.0 verfügbar, die irgendwann auf dem Markt erscheinen soll.

Die Installation des Programms Winbase gestaltet sich sehr einfach. Sie legen die gelieferte Diskette in das Laufwerk A: und geben den Befehl »SETUP« ein. Nach Eingabe des Ziellaufwerks und -verzeichnisses läuft die Installation automatisch ab. Um den Start von Winbase beim Anklicken von DBF-Dateien zu automatisieren, müssen Sie die Datei WIN.INI ändern. Sie fügen dazu folgenden Eintrag ein, der übrigens im Handbuch falsch beschrieben ist:

```
DBF = Winbase.EXE^.DBF
```

Mit Hilfe dieses Eintrags wird Winbase automatisch geladen, sobald Sie eine DBF-Datei anklicken. Eine oder mehrere geöffnete Datenbanken unter Winbase stellen sich so auf dem Bildschirm dar, wie es *Bild 2* zeigt.

Was und wie läßt sich mit Winbase arbeiten?

Solange das automatsiche Laden der Datenbanken nicht in WIN.INI festgeschrieben ist, benutzen Sie die Pull-Down-Menüs von Winbase und eine Maus, um eine Datenbank zu laden und zu bearbeiten. Mit Hilfe der vorhandenen Pull-Down-Menüs von Winbase können Sie:

• eine Datenbank öffnen, Suchlisten erstellen, Daten drucken und Hilfefunktionen abrufen (Menü Datei);

• auf die Ablage von MS-Windows (Clipboard) zugreifen, Datensätze als gelöscht markieren und zufügen, Felder aktualisieren, Memo-Felder in einem eigenen Fenster ansehen und ändern (Editieren-Menü);

• Datensätze in auf- und absteigender Reihenfolge sortieren (eins oder mehrere Felder), Bedingungen definieren und diese Definitionen als Datei (QEF) speichern, nur gelöschte oder alle Datensätze anzeigen (Auswahl-Menü);

• Informationen in Datensätzen suchen, in einzelnen oder allen Feldern suchen, Datensätze über die Datensatznummer direkt anspringen (Suchen-Menü);

• Spalten und ihre Position verändern, numerische Felder summieren und statistische Daten separat anzeigen lassen (Layout-Menü).

Eine Besonderheit findet sich bei Winbase, wenn es um das Formulieren von Bedingungen zur Darstellung von Datensätzen auf dem Bildschirm geht. Neben der Möglichkeit, die üblichen boolschen Operatoren einzusetzen, bietet Winbase noch sogenannte Vergleichszeichen (*Wie* und *Nicht Wie*) an, die bei der Definition von Bedingungen hilfreich sind. Beim Einsatz der Bedingung *Wie*, können die von MS-DOS her bekannten Jokerzeichen (Wildcards) benutzt werden.

* steht für 0 oder mehr zutreffende Zeichen
? steht für ein einzelnes, zutreffendes Zeichen

Allerdings muß trotz aller Freude über diese neuen Möglichkeiten erwähnt werden, daß das Programm nicht zwischen Groß- und Kleinschreibung unterscheidet.

Arbeiten mit dem Winbase Definer

Mit dem Definer können Sie neue dBase-Datenbanken anlegen. Dabei werden die von dBase her bekannten Feldtypen unterstützt:
C = Zeichenfelder (max. 254 Stellen)
N = Numerische Felder (max. 19 Stellen)
D = Datum Felder (max. 8 Stellen)
L = Logische Felder (max. 1 Stelle)
M = Memo Felder (max. 4096 Stellen)

Beim Einrichten von Datenbanken orientiert sich der Definer an seinem Vorbild dBase. Die erweiterten Möglichkeiten wie zum Beispiel 1028 Felder, 64 Kbyte lange Memo-Felder usw. beim Clipper und Foxbase können hier nicht genutzt werden. Bei der Einrichtung eines Feldtyps werden die Standardwerte automatisch vorgeschlagen. Sie können diese Werte einfach übernehmen oder eigene, andere Werte eintragen. Die unterstützen Dateiformate sind:
Datenbank = .DBF
Memo = .DBT

Datenbanken können wie unter dBase maximal eine Milliarde Datensätze aufnehmen. Jeder

Datensatz darf dabei höchstens 4000 Byte enthalten, die auf nicht mehr als 128 Felder verteilt sind.

Memodateien können maximal 4096 Zeichen lang sein. Jedes Feld ist in Blöcke zu 512 Byte aufgeteilt. Bei Eingabe von zum Beispiel 513 Byte in ein Memofeld, belegt die Memodatei 1024 Byte auf der Festplatte. Das deutet nicht gerade auf ökonomischen Umgang mit den Speicherkapazitäten hin.

Besonders interessant für Anwender, die nach wie vor mit dem bewährten und fehlerfreien dBase II arbeiten ist, daß die mit dem Definer erstellten Datenbanken auch im dBase II-Format abgelegt werden können.

Zwar lassen sich Datenbanken problemlos kopieren, indem Sie einfach eine Datenbank unter einem neuen Namen ablegen, aber der Import von Daten aus einer anderen Winbase-Datenbank ist nicht als eigenständiger Menüpunkt vorgesehen.

Die Struktur der Datenbank läßt sich im Definer verändern. Allerdings muß man dabei darauf achten, daß die bereits gespeicherten Daten in der Datenbank verstümmelt werden können oder ganz verlorengehen. Nur im Definer ist es möglich, zum Löschen markierte Datensätze zu »packen«, das heißt: physikalisch zu löschen und auch physikalisch zu sortieren. Beim Löschen werden die Daten aus der Datenbank entfernt und sind nicht wieder herstellbar. Beim Sortieren werden unter recht hohem Zeitaufwand die Datensätze in der Datenbank in eine neue Reihenfolge gebracht. Sie stehen dann nicht mehr in der Reihenfolge der Eingabe in der Datenbank, sondern entsprechend den angegebenen Sortierkriterien. Beim Sortieren ist es möglich, auch mehrere Felder gleichzeitig zu sortieren. Die Druckmöglichkeiten des Definer lassen Sie die Struktur der Datenbank auf den Drucker oder auch in eine Datei ausgeben.

Thomas Langhammer

QuickPascal 1.0:

Systemnahe Programmierung mit QuickPascal

Mit Interrupt-Aufrufen können PCs auf Systemebene effektiv programmiert werden. Dies bleibt jedoch häufig Systemprogrammierern und denjenigen Insidern vorbehalten, die Assembler-Sprachen im Schlaf beherrschen.

Dabei ist jetzt mit QuickPascal das Aufrufen von Interrupts sowie die Manipulation einzelner Speicherzellen oder gar Bits außerordentlich komfortabel möglich.

Der QuickPascal-Programmierer braucht lediglich zu wissen, welcher Interrupt für welchen Zweck vorgesehen ist und welche Werte den Registern übergeben werden müssen. Alles andere – das Ablegen von Offset- und Segmentadressen auf dem Stack sowie das Sichern des Flagregisters – nimmt ihm der Compiler ab.

Systemprogrammierung ohne Assembler

Diese Möglichkeiten des QuickPascal-Compilers sowie die leichte Erlernbarkeit der QuickPascal-Syntax tragen in hohem Maße dazu bei, daß relativen Neueinsteigern wie z.B. Schülern im Informatikunterricht die Systemprogrammierung demonstriert werden kann.

Hier wird nun an einigen einfachen Beispielen gezeigt, wie komfortabel die Handhabung von Software-Interrupts mit QuickPascal auch ohne Assemblerkenntnisse ist. Es handelt sich bei den Demos um Funktionen der Software-Interrupts 13h zur Behandlung des Diskettencontrollers und 10h zur Manipulation der Cursorgröße. Außerdem liest das Beispielprogramm die Diskettenparametertabelle, die über den Interrupt 1Eh zu erreichen ist.

```
DISKPARA liest und ändert die Diskettenparametertabelle mit QuickPascal 1.0
 Aktuelle Diskettenparameter:           neue Parameter:    mögliche Werte

 Steprate-Byte       (ms):  6           6                  2, 4 oder 6
 Motor-Nachlauf (sec/18):   37          37                 0 ... 255
 Byte je Sektor:            2           2                  fix
 Sektoren je Spur:          15          15                 fix
 Init-Byte für FORMAT:      ÷           ÷                  beliebiges Zeichen
 Kopfberuhigung      (ms):  15          15                 0 ... 255
 Motor-Anlauf     (sec/8):  2           2                  0 ... 255

 Sollen die Parameter geändert werden ?  j/n :

   (c) 1989  F-J Steiner
```

Flexible Interrupts

Interrupts sind aus dem Konzept von PCs nicht wegzudenken. Sie erst ermöglichen eine hohe Flexibilität des Betriebssystems: eine Routine, die entweder neu geschrieben oder nur im Speicher verschoben wird, kann mit dem »alten«, d.h. unveränderten Interrupt aufgerufen werden, da sich die Position des Zeigers, der auf die Routine weist, nicht verändert hat.

Grundsätzlich muß man wissen, daß der Aufruf eines Interrupts die CPU veranlaßt, den Pro-

grammlauf zu unterbrechen. Nach der Sicherung der Adressen des unterbrochenen Programms errechnet die CPU aus der Interrupt-Nummer die Speicherzellen, in denen die Startadresse für das auszuführende Programm abgelegt ist. Nach einem Sprung an die gefundene Adresse wird das Programm ausgeführt und anschließend wieder zum unterbrochenen Programm zurückgekehrt. Dies gilt für die meisten Interrupts. Es gibt jedoch drei Ausnahmen: die Tabellen, die keinen ausführbaren Programmcode enthalten. Die Diskettenparametertabelle zum Beispiel, deren Startadresse über den Interrupt 1Eh zu erreichen ist, enthält anstelle von Maschinenanweisungen diverse Einstellparameter für den Diskettencontroller. Weil ein Sprung der CPU an die Startadresse der Tabelle den Abschied ins Rechnernirwana bedeuten würde, benötigt ein solcher Interrupt eine vorsichtige Handhabung: Er darf vor allem nicht aufgerufen werden!

Doch zurück zu den Software-Interrupts, die lauffähige Routinen starten können. Die meisten dieser Interrupts besitzen eine Vielzahl von Funktionen. Jede Funktion dient dabei einem anderen Zweck: Der Interrupt 10h beispielsweise, der die Ein- und Ausgabeoperationen des Bildschirms steuert, enthält eine ganze Bibliothek von Funktionen, die z.B. den Videostatus oder das Zeichen an der Cursorposition ermitteln sowie die Cursorgröße oder Farben verändern und vieles mehr.

Parameterübergabe an Register

In der Regel erwartet eine Interrupt-Routine, daß einzelne Register mit bestimmten Werten gefüllt sind. Die Routine wertet den Inhalt aus und wählt z.B. aufgrund des Inhalts von Register AH die gewünschte Funktion.

QuickPascal stellt zur Datenübergabe an Interrupt-Routinen einige nützliche Datenstrukturen zur Verfügung. Die Unit `DOS`, die mit dem Schlüsselwort `Uses` aufgerufen werden muß, enthält einen `Record` mit dem Namen `Registers`, über den sich alle Register komplett oder nur deren oberes oder unteres Byte ansprechen lassen:

```
TYPE
  Registers = RECORD
      CASE Integer OF
      0 : (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Word);
      1 : (AL, AH, BL, BH, CL, CH, DL, DH : Byte);
    END;
```

Wenn Sie eine Variable `reg` vom Typ des Records `Registers` definiert haben, können Sie eine Zuweisung in der folgenden Form durchführen, wie sie auch in der Prozedur `disk_init` des Beispielprogramms gezeigt ist:

```
reg.ah := 0;
reg.dl := 0;
```

Die Werte können dezimal – wie dargestellt – oder hexadezimal mit vorangestelltem »$« übergeben werden. Für den Interrupt 13h, der für Disketten- und Festplattenzugriffe zuständig ist, haben die Parameter folgende Bedeutung: Der Wert »0« im Register AH steht für die Funktionsnummer 0 (Initialisierung des Controllers); die »0« im Register DL für das Ziel der Aktion, den Diskettencontroller.

Der Aufruf des Interrupts erfolgt nach der QuickPascal-Syntax in der allgemeinen Form

```
PROCEDURE Intr(  interrupt_number : Byte;
            VAR register_values  : Registers )
```

Für das Beispiel `disk_init` lautet der Aufruf dann

```
intr($13,reg);
```

Nach dem gleichen Prinzip erfolgt der Aufruf des Software-Interrupts 13h zur Änderung der Cursorgröße. Im Programm wird der Cursor auf die Zeilen (je Zeichen) 0 bis 13 vergrößert. Durch diese Manipulation erscheint der Cursor als heller Block. Nach Beendigung der Änderungsprozedur `aendere_tab` wird durch einen weiteren Interrupt-Aufruf, allerdings mit anderen Registerinhalten, der Cursor auf einen schmalen Strich zurückgesetzt.

Interrupts als Tabellenzeiger

Die Diskettenparametertabelle liegt als Block von 11 Byte Länge ab Adresse 4 * 1Eh = 78h im ersten Segment des Arbeitsspeichers.

Sie enthält zwölf Parameter, die die Arbeitsweise des Diskettencontrollers beeinflussen:

Byte	Zweck
1/2	Steprate
2/2	Rückstellverzögerung für Schreib/Lesekopf
2	DMA-Modus
3	Motor-Nachlaufzeit
4	Byte je Sektor
5	Sektoren je Spur
6	log. Intervall zwischen Sektoren
7	Datentransferlänge (DTL)
8	phys. Intervall zwischen Sektoren
9	Formatierungszeichen
10	Ruhezeit des Schreib/Lesekopfes
11	Motor-Anlaufzeit

Lärmenden Floppies mit Bits und Bytes auf den Leib gerückt

Wer sich schon häufiger über seine lärmenden Floppylaufwerke geärgert hat, kann das Übel jetzt an der Wurzel packen. Werden die Werte, die direkten Einfluß auf Laufwerksmotoren und Schreib/- Leseköpfe haben, vorsichtig verändert,

kann dies die Geräuschentwicklung nachhaltig verringern und damit auch die Lebensdauer der Laufwerke verlängern. Außerdem sind spürbare Geschwindigkeitssteigerungen bei Diskettenzugriffen festzustellen.

Das Programm zeigt einen Ausschnitt der Laufwerkstabelle und läßt die Manipulation der relevanten Bytes zu. Die zulässigen Werte werden jeweils rechts auf dem Bildschirm angezeigt.

Ausblick

Natürlich lassen sich mit QuickPascal nicht nur schon vorhandene Software-Interrupts aufrufen. Auch die Deklaration einer selbstgeschriebenen Prozedur als Interrupt ist möglich. Auf diese kann dann ein Interrupt-Zeiger »umgebogen« werden.

An dieser Stelle sollte jedoch zunächst gezeigt werden, wie einfach der Einstieg in die systemnahe Programmierung mit QuickPascal gelingen kann. Die wirklich alles umfassende Hilfestellung der Entwicklungsumgebung macht den Einsatz der vorgegebenen Datenstrukturen und Prozeduren nahezu zum Kinderspiel.

Franz-Josef Steiner

```
{ ——————————————————————————————————————————————————————
 DISKPARA.PAS  /  09.89  /  F-J Steiner
 Liest die Diskettenparametertabelle (Interrupt 1Eh)
 und verändert sie nach Bedarf
 —————————————————————————————————————————————————————— }

program diskpara;

   uses crt, dos;

{ ——— Globale Variablen ——————————————————————————————— }

   var  nachlauf, anlauf,
        stepbyte,
        initbyte, ruhe,
        byte_sektor, sektor_spur : byte;
        eingabe, steprate        : char;

{ ——— Zuweisung der Tabellenadresse —————————————————— }

        offs : integer absolute $0:$1E*4;     { Offset-Adresse  }
        segm : integer absolute $0:$1E*4 + 2; { Segment-Adresse }

{ ——— Prozedurdeklarationen ——————————————————————————— }

procedure disk_init;
{ ===  führt einen Reset des Diskettencontrollers durch  === }
  var reg : registers;
begin
      reg.ah := 0;            { Funktion 0h }
      reg.dl := 0;            { Kennwert für Diskettenlaufwerk }
      intr($13,reg);          { Aufruf von Interrupt 13h }
end;

procedure cursor(oben, unten : byte);
{ ===  ändert die Größe des Cursors  === }
  var reg : registers;
begin
       reg.ah := 1;           { Funktion 1h }
       reg.ch := oben;        { Cursorrand oben  }
       reg.cl := unten;       { Cursorrand unten }
       intr($10,reg);         { Aufruf von Interrupt 10h }
end;

procedure lies_tab;
{ ===  liest Ausschnitte der Parametertabelle
       und übergibt die Werte an Variablen      ===  }

begin
       stepbyte    := mem[segm:offs];
               if stepbyte = 223 then steprate := '6'
                  else if stepbyte = 239 then steprate := '4'
                  else if stepbyte = 255 then steprate := '2';

       nachlauf    := mem[segm:offs+2];
       byte_sektor := mem[segm:offs+3];
       sektor_spur := mem[segm:offs+4];
       initbyte    := mem[segm:offs+8];
       ruhe        := mem[segm:offs+9];
       anlauf      := mem[segm:offs+10];
end;

procedure zeige_tab;
{ === stellt die übergebenen Werte am Bildschirm dar  === }

begin
       gotoxy(3,3);
       highvideo;
       write('Aktuelle Diskettenparameter:');
       normvideo;
       gotoxy(3,6);
       write('Steprate-Byte      (ms):  ',steprate);
       gotoxy(3,8);
       write('Motor-Nachlauf (sec/18):  ',nachlauf);
       gotoxy(3,10);
       write('Byte je Sektor:           ',byte_sektor);
       gotoxy(3,12);
       write('Sektoren je Spur:         ',sektor_spur);
       gotoxy(3,14);
       write('Init-Byte für FORMAT:     ');
       write(chr(initbyte));
       gotoxy(3,16);
       write('Kopfberuhigung     (ms):  ',ruhe);
       gotoxy(3,18);
       write('Motor-Anlauf    (sec/8):  ',anlauf);
end;

procedure lies_byte(x,y : byte; var wert : byte; max : byte);
{ ===  Einlesen von Byte-Variablen von der Tastatur  ===
  Eingaben:    Bildschirmkoordinaten:  x, y
               Zu änderndes Byte:      wert
               Maximal zul. Wert:      max
  Ausgaben:    Veränderter Wert  ====================== }
       var eing_str  : string[3];
           fehler,i  : integer;
           ch        : char;
begin
       repeat
               gotoxy(x,y); write(wert);
               gotoxy(x,y);
               fehler := 0;
               eing_str := '';
               readln(eing_str);
               if length(eing_str) > 0 then
                     val(eing_str,wert,fehler);
       until (fehler = 0) and (wert >= 0) and (wert <= max);
end;

procedure lies_char(x,y : byte; var wert : char);
{ ===  Einlesen von Char-Variablen von der Tastatur  ===
  Eingaben:    Bildschirmkoordinaten:  x, y
               Zu änderndes Zeichen:   wert
  Ausgaben:    Veränderter Wert  ====================== }
       var ok  : boolean;
begin
       repeat
               ok := false;
               gotoxy(x,y); write(wert);
               gotoxy(x,y);
               eingabe := readkey;
               if eingabe = #13 then ok := true
                 else if eingabe in ['!'..''] then
                            begin
                                  wert := eingabe;
                                  ok := true;
                            end;
               gotoxy(x,y); write(wert);
       until ok;
 end;

procedure aendere_tab;
{ ===  Ändert die Parametertabelle  === }
       var initchar    : char;
begin
       cursor(0,13); gotoxy(40,3); highvideo;
       write('neue Parameter:');
       gotoxy(60,3);
       write('mögliche Werte');
       normvideo;
       gotoxy(60,6);
       write('2, 4 oder 6');
       gotoxy(60,8);
       write('0 ... 255');
       gotoxy(60,10);
       write('fix');
       gotoxy(60,12);
       write('fix');
       gotoxy(60,14);
       write('beliebiges Zeichen');
       gotoxy(60,16);
       write('0 ... 255');
       gotoxy(60,18);
       write('0 ... 255');
       repeat
               lies_char(40,6,steprate);
       until steprate in ['6','4','2'];
               if steprate = '6' then stepbyte := 223
                  else if steprate = '4' then stepbyte := 239
                  else if steprate = '2' then stepbyte := 255;

       lies_byte(40,8,nachlauf,255);
       gotoxy(40,10);  write(byte_sektor);
       gotoxy(40,12);  write(sektor_spur);
       initchar := chr(initbyte);
       lies_char(40,14,initchar);
       initbyte := ord(initchar);
       lies_byte(40,16,ruhe,255);
       lies_byte(40,18,anlauf,255);
       mem[segm:offs]   := stepbyte;
       mem[segm:offs+2] := nachlauf;
       mem[segm:offs+8] := initbyte;
       mem[segm:offs+9] := ruhe;
       mem[segm:offs+10] := anlauf;
       cursor(6,7);
end;

{ ——————————————————————————————————————————————————————— }
{             Haupt - Programm                             }
{ ——————————————————————————————————————————————————————— }
begin
       clrscr;                          { Bildschirmaufbau }
       textcolor(0); textbackground(7);
       clreol;                          { inverse Darstellung }
       write(' DISKPARA  liest und ändert die Disketten',
             'parametertabelle  mit QuickPascal 1.0');
       gotoxy(1,24);
       clreol;
       write('    (c) 1989  F-J Steiner');
       textcolor(7); textbackground(0);
       lies_tab;                        { Tabelle lesen }
       zeige_tab;                       { und darstellen }
       gotoxy(3,21); highvideo;
       write('Sollen die Parameter geändert werden ?  j/n : ');
       normvideo;
       eingabe := readkey;
       if eingabe in ['j','J'] then    { Tabelle ändern, wenn }
              begin                     { gewünscht; nach der  }
                      aendere_tab;      { Änderung den Disketten-  }
                      disk_init;        { controller initialisieren }
              end;
       gotoxy(1,24);
end.
```

Hintergrundinformationen:

Grundlagen dezentraler Datenverarbeitung mit LAN-Manager/ LAN-Server-APIs

PC-Netzwerke sind reifer geworden: Sie haben sich von ihrer ursprünglichen Aufgabe, einen gemeinsam nutzbaren Festplatten- und Druckerservice zu bieten, zur Basis unternehmensweiter Informationssysteme entwickelt.

Um die Rolle eines firmenweiten Informationssystems zu übernehmen und mit existierenden Minicomputern und Mainframe-Systemen konkurrieren zu können, muß die vernetzte Datenverarbeitung zwei Anforderungen erfüllen:
1. PC-Netzwerke müssen mehreren Anwendern die gemeinsame Nutzung von Informationen auf effiziente, zuverlässige und einfache Weise ermöglichen.
2. PC-Netzwerke müssen eine einfachere umfassendere Administration und ausreichende Sicherheitsvorrichtungen bieten.

Eine wichtige Voraussetzung für die Erstellung von Netzwerk-Applikationen ist das Applications Programming Interface (API) für den IBM LAN-Server und den Microsoft LAN-Manager.

Client/Server-Architektur

LAN-Manager-APIs unterstützen die Entwicklung einer völlig neuen Generation von Applikationen, die auf der Client/Server-Architektur (dezentrale Datenverarbeitung) basieren. Die dezentrale Datenverarbeitung unterteilt eine Applikation in eine Workstation- (front-end) und in eine Server-Komponente (back-end). Der Arbeitsplatzrechner sendet Informationen an den Server und erhält von ihm Antworten. Server werden damit zu aktivieren und intelligenteren Teilnehmern, die nicht mehr länger als eine Art »Verkehrspolizisten« nur die gemeinsame Nutzung von Dateien und Druckern regeln.

Eines der bekanntesten Beispiele einer modernen Client/Server-Architektur ist der Ashton-Tate/Microsoft SQL-Server (Structured Query Language). Er arbeitet als eine leistungsfähige Back-end-Datenbank-Maschine, deren Funktionalität Mainframe-Datenbanken wie DB2 ähnelt. Front-end-Client-Applikationen senden SQL-Abfragen jetzt über das Netzwerk an den SQL-Server, ohne die Dateien direkt im Server zu manipulieren. Der SQL-Server verarbeitet die Anfrage und gibt das Ergebnis an die Workstation zurück. Ferner bearbeitet der SQL-Server alles, was das Suchen oder Sortieren von Dateien, Vorgangsbearbeitung, Datenbank-Zugriff, Datenbank-Sicherheit, Daten-Sicherheitsprüfung usw. betrifft. Bis jetzt haben bereits über 30 Software-Hersteller, u.a. Ashton-Tate, Borland, Dataease, Information Builders, Microsoft und Revelation Technologies, erklärt, daß sie ihre Arbeitsplatzrechner-Applikationen mit Schnittstellen zum SQL-Server ausstatten werden. Weitere Beispiele für eine Client/Server-Architektur:

• Der DCA/Microsoft Communications Server bietet Netzwerk-Arbeitsplatzrechnern IBM SNA-Dienste, wie 3270-Terminal- und Printer-Emulation, Dateitransfer, direkte Programm-zu-Programm-Kommunikation (APPC) sowie asynchrone Terminal-Emulation.

BEI NETZWERKEN SIND WIR FÜR ALLE OFFEN...

4th Dimension
Abdobe II
AutoCAD
Clipper
Crosstalk XVI
Dataease 2.5
DataFlex
dBASE III Plus
dBASE IV
DBXL
EasysGate
Euroscript LAN
F & A
FoxBase+/386
FoxBase+/LAN
Framework III
Freehand
Freelance Plus 3.0
GEM Desktopc
GEM Draw
GEM Graph
GRAPHWRITER II
Harvard Graphics
Harvard Total Project Manager
Hypercard
Informix-4GL
Informix-ESQL/C
Informix-SQL
Javelin
Lotus 1-2-3/2.01
Lotus 1-2-3/3.0
MacDraw
MacProjekt
MacWrite
MANUSCRIPT 2.0
Metro
MS Assembler
MS C Compiler
MS Chart
MS Excel
MS Multiplan 3.0
MS Project Netz
MS Rbase System
MS Word 4.0
MS Word for Macintosh
MultiMate Advantage II

NETBase
OMNIS 3 Plus
Open Access II
Professional ORACLE Version 5.1B
ORACLE Application Tools V5.1B
PageMaker for the Macintosh
PageMaker for the PC
PFS:Professional
Plan Perfect Netz
Poly-STAR/220 & 240
Quattro
Q & A

Reflex
Samna Word Plus IV
Sidekick Plus
Smart System 3.1
Sprint
SQLBase
Supercalc 4 Netz
SuperPaint
Superproject Expert LAN
Symphony 2.0
Tex-Ass Windows
The Smart Software System
Time Line
Turbo C
Turbo Pascal
Turbo Prolog
Ventura Publisher
Volkswriter III LAN
Windows/386
Word Perfect Netz
WordStar 2000
WordStar 4.0

SYSTEMS 89
Halle 18, Stand C3
Freigelände vor Halle 18 - 7

3+ Open LAN Manager ist ein für alle offenes Netzwerk-Betriebssystem. Es verbindet die Welten von DOS-, Macintosh-, OS/2-, UNIX-, DEC- und IBM-Großrechnern. Die Rechnerleistung kann optimal zwischen Server und Arbeitsplatz verteilt werden.

3+ Open LAN Manager ist kompatibel zu Protokoll-Standards von DEC, IBM, Xerox und anderen.

Wirksame Datenschutz- und Management-Funktionen sowie zahlreiche Programmierschnittstellen beweisen Flexibilität und Leistungsfähigkeit des 3+ Open LAN Managers.



3Com

Netzwerklösungen mit System.

Gustav-Heinemann-Ring 123, D-8000 München 83

3Com-Partner in Deutschland

3Com GmbH, Gustav-Heinemann-Ring 123, 8000 München 83
3Com GmbH, Frankfurter Str. 33–35, 6236 Eschborn
3Com GmbH, Braunfelder Chaussee 216, 2000 Hamburg 71

Distributoren:
Adcomp GmbH, München. Atlantik Elektronik, Martinsried. ComputerLinks GmbH, München. Computer 2000 AG, München. Positronika GmbH, Essen.

Value Added Reseller:
CE Computer Equipment GmbH, Bielefeld. Steckenborn Computer, Gießen. Interface Concilium, München. Echtzeit Computer, Göttingen. Zinck, Neuburg a. d. Donau. C.V.B., Bonn. CENET München, Germering.

Autorisierte 3Com-Händler:
Berlin: Compunet Berlin, Dr. Dohrenberg GmbH, Horn & Görwitz, TCV Text-Computer Vertr. GmbH, ABAC Datentechnik GmbH, Pandasoft, Inline Software
Hamburg: Data Equipment GmbH, P.C.P. Pfalzgraf, RK Datensysteme, MCT Microcomputer Team GmbH, Microware Hamburg, G.A.I. Hamburg. BPO Computerladen, Pinneberg. Cebus GmbH, Kiel. Nagel & Knack, Kiel. Reese, Kiel. TSS Stoltenberg GmbH, Ahrensburg. Selck Computersysteme, Flensburg. Ahnemann & Kunze, Bremen
Hannover: EDV-Beratung Petra Schulze, PC-Systems, Garbsen. B.I.V.G., Laatzen. Mesob Software, Braunschweig. Computerland, Göttingen
Düsseldorf: ACC, Brosius & Köhler, Data Service Düsseldorf, Microware Düsseldorf, Schasiepen, Experteam, Duisburg. Rennen, Duisburg. Rennen, Essen. Bcsc, Essen. Middelberg u. Göx, Lotte-Büren. Experteam, Dortmund. Bongartz und Schmidt, Bochum. Laufenberg, Bochum. AKA-EDV, Bochum. RWL Computersysteme GmbH, Unna
Köln: Experteam Köln, Micro Cosmos, PCS Grönwoldt & Kempfer, ICT, Aachen. Annotext GmbH, Düren. Cosmoconform, Hennef-Heisterschoss. Heydweiller & Krebs, Bonn. Visarius GmbH, Bonn. Hennefeld, Wiesbaden. Lanware GmbH, Montabaur. Röth Datensysteme, Wuppertal
Frankfurt: Data Service Rhein-Main, Henneveld Frankfurt, Microware Frankfurt, S & B Software GmbH, Dreieich. INS GmbH, Friedrichsdorf. Linke EDV, Driedorf. HCS GmbH, Rödermark. C.O.S. Computersysteme, Saarbrücken. Gecomp GmbH, Mannheim. Data Link, Wiesloch
Stuttgart: ML Software, Adlon OHG, Ravensburg. BIWISI GmbH, Bernhausen. CEB Böblingen. Data Link, Leonberg. Data Service, Karlsruhe. C-Tronic GmbH, Karlsruhe. Pfotenhauer, Achern. Comformatik, St. Georgen. Computerland, Friedrichshafen. Bercher GmbH, Tettnang
München: Aavalon, BCR Vertriebsgesellschaft mbH, CE-Plus, Datalog Software, Didas Computer, Franklin Computersysteme, Ingenieur Ring Bittner & Krull, Micro Automation Consult, Microage München, PC-Plus GmbH, Portasoft, Singhammer, Softing GmbH, Servonic, Ergo Software, CNS Computer Network Systems, Columbus Datentechnik, Desk GmbH, Ottobrunn. CENET München, Germering. Christoph Seitz GmbH, Pullach. ABS Computerland, Polling. Megabyte, Oberschneiding. WDV GmbH, Garching. Microware München, Ismaning. MST GmbH, Neubiberg. Microage, Nürnberg. Computer Martin, Würzburg. Koda GmbH, Würzburg. B.V.S. Datensysteme Stadtbergen.

ECO C - TOOLS

DIALINT
schnelle Windows im Text- und Graphik-Modus. Durch Interpretation von Text-Dateien o. compiliert

AUTODIAL
Eine Dialog- und Aktionsmaschine das User Interface Management System von ECO

GRAPHIK
Grapik-Funktionsbibliothek

GEOMETRIE
Funktionen nach CNC Standard

DATENSTRUKTUREN
C-Implementierung abstrakter Datenstrukturen, Sortierverfahren etc.

DOS und BIOS-Funktionen
Zugriffsroutinen

STRUKTOR
Druckreife Struktogramme für C, Pascal etc. und Pseudo-Code, Kommentar-Verarbeitung, Konfigurierbare Ausgabe

Demo-Disketten erhältlich !

ECO SCHULUNGEN

C-Trainer
Lernkurs mit C-Interpreter für Autodidakten (auch Quellcode erhältlich)

C-SEMINARE
Über unsere Produkte und Ihre Anwendungsprobleme



für Electronische Communication und Organisation GmbH

...deutlich mehr als nur Software!

Abt. V 2
Landshuter Straße 37
D-8400 Regensburg 1
Tel. 0941/71098
Fax: 0941/74833

BKS-TOOLWARE sind High Tech 'C'-Tools und ...

- **BKS-ISAM** die schnelle und sichere Datenbank
- **BKS-WINDOW** Bildschirmdialoge und Fenstertechnik leicht gemacht
- **BKS-LISTER** für Listenerstellung und Druckeranpassung
- **BKS-GRAPH** GKS-Grafik-Standard-Programmierung z.B. für EGA, CGA, VGA, Plotter und diverse Matrix-Drucker
- **BKS-GEOMETRIE** mathematische Berechnungen z.B. Schnittpunkte, Tangenten und Kreise

Lieferbar für MS-DOS, OS/2, SCO XENIX 286/386, UNIX V/386, UNIX V, VMS und FlexOs (auch spezielle Cross-Development-Pakete).

... Training für ...

- **'C'** Intensiv-Seminare für Einsteiger, Umsteiger und Experten
- **SCO XENIX** System-Administration und Shell-Programmierung
- **BKS-Produkte** Praxis-Seminare für die Anwendungsprogrammierung

... produktive Software-Entwicklung!

Besuchen Sie uns bald auf der SYSTEMS 89 München 16.-20.10.'89 Halle 7 · Stand A 1

INFORMATIONS-COUPON
Bitte schicken Sie mir Informationen zu:
☐ Produkte ☐ Betriebssysteme
☐ Schulungsprogramme
Absenderangabe .... und ab geht die Post!

Guerickestr. 27
1000 Berlin 10
Telefon: (030) 342 30 66-67
Telefax: (030) 342 84 13

BKS Software

• Eine dezentrale Applikation von Seros Corporation steuert die gemeinsame Nutzung von PC-Informationen in Netzwerken. Sie bietet die Möglichkeit, gemeinsame Informationen wie Textdateien, Kalkulationstabellen, Publikationen und Zeichnungen aufzufinden und zu verwalten. Microsoft Windows-gesteuerte Workstation-Software vereinfacht das Suchen von Informationen, die auf vielen Servern in einem Netz verteilt sind. Der Suchvorgang stützt sich auf beschreibende Attribute, wie Anwendername, Objekt, Thema und Datei. Die Server-Software erstellt einen Informationskatalog über Objekte, die vom System bearbeitet werden, und führt Prüf- und Selbstdiagnose-Verfahren sowie die Versionsverwaltung durch.

• Das Xcellenet Wide Area Network (WAN) Management System unterstützt eine Gruppe vernetzter LAN-Manager/LAN-Server-Leitrechner für die breite Vernetzung von vielen PCs unter MS-DOS und MS OS/2 Systemen über asynchrone Fernmeldeleitungen. Die Server-Rechner können ebenso mit Großrechnern verbunden werden, wobei die Großrechnersysteme als virtuelle Leitrechner (File-Server) im WAN erscheinen. Das Xcellenet Server-System liefert praktisch unbegrenzte Arbeitskapazität. Um die Kapazität des Systems zu erhöhen, können zusätzliche LAN-Manager/LAN-Server-Leitrechner hinzugefügt werden. Außerdem führt der Einsatz von mehr als einem Server zu einer Lastverteilung, also einem optimalen Einsatz von Netzwerk-Ressourcen und Systemredundanz für den Fall, daß ein Server ausfällt.

Vorteile der Client/Server-Datenverarbeitung

Client/Server-vernetzte Applikationen bieten unter den Aspekten Leistung, Sicherheit, Administration und Rentabilität eine Reihe von Vorteilen:

Die Leistungssteigerung von Client/Server-vernetzten Systemen führt zu einer einfacheren, effizienteren und zuverlässigeren gemeinsamen Nutzung von Informationen durch einzelne Benutzergruppen.

Einer der wichtigsten Gründe für die erhöhte Leistung ist, daß die Server-gesteuerte Software unter dem neuen Betriebssystemstandard MS OS/2 läuft. MS OS/2 erfordert von Server-Applikationen einen außergewöhnlich großen Speicherbedarf, Multitasking-Fähigkeit und einen umfangreichen API-Satz. Als Server dienen die leistungsfähigsten Computer in einem Netz. Da Server-gesteuerte Prozesse auf diese Computer zugreifen können, kommen deren Leistungsvorteile allen Benutzern im Netz zugute.

Die Client/Server-Architektur führt zu einer beträchtlichen Reduzierung des Netzwerk-Verkehrs, weil die Server-gesteuerten Applikationen auf den Rechnern laufen, die auch Benutzerdateien oder andere Ressourcen enthalten. Da die Server-Applikationen auf die gesamten Server-Daten lokal zugreifen, ist es nicht erforderlich, große Blöcke von Informationen über das Netz zu den Workstations zu transferieren. Nur ganz spezielle, von einem Arbeitsplatzrechner angeforderte Informationen, werden über das Netzwerk gesendet.

Ein dritter Vorteil, den die Client/Server-Architektur bietet, besteht darin, daß die Datensatz-Sperrlogik der Workstation-Applikationen überflüssig wird, weil die Server-gestützte Applikation die einzige ist, die tatsächlich auf die Daten direkt zugreift. Der Server kann Parallel-Zugriffssteuerung durch andere, effektivere Mittel bieten. Zum Beispiel nutzt der SQL-Server einen hochentwickelten Page Level-Sperrmechanismus mit verschiedenen Sperrebenen. Auf diese Weise kann er die Anforderungen vieler Anwender-Gruppen, die alle auf die gleiche Datenbank zugreifen, erfüllen und einen durch Parallelzugriff verursachten Stillstand verhindern.

Ein weiterer großer Leistungsvorteil der Client/Server-Architektur besteht darin, daß Server-gesteuerte Applikationen relativ einfach für besondere Aufgaben optimiert werden können. Ein Server-Prozeß kann zum Beispiel Ablaufsteuerungen und globale Optimierungen, wie die Verarbeitung von Anfragen in der effizientesten Reihenfolge, durch entsprechende Steuerung der Schreib-/Lesekopf-Bewegung oder mittels Zugriff auf den Cache-Speicher, ausführen. Auch die Verarbeitung von Arbeitsplatzrechner-Anforderungen parallel nach einem optimierten Dispositionsalgorithmus ist möglich. Dieses Prinzip bietet den System-Benutzern eine wesentlich schnellere Antwortzeit.

Die dezentrale Datenverarbeitung verbessert die Netzwerk-Sicherheit und die Systemadministration. Einer der Vorteile der Server-Dienste besteht in der Möglichkeit, die Zugriffsberechtigung des Benutzers und den Datenzugriff an einer zentralen Stelle, dem Server, zu überprüfen. Beim SQL-Server bedeutet dies, daß jede Anforderung, unabhängig von deren Ursprung, zuerst in einer Benutzerprofil-Datenbank, die Teil der Server-Software ist, auf ihre Gültigkeit hin überprüft wird. Dadurch kann festgestellt werden, ob der Benutzer die entsprechende Zugriffsberechtigung hat.

Außerdem wird jede Datenfortschreibungs- oder Datenmodifikations-Anforderung an den SQL-Server anhand einer Liste von Datensicherheitsregeln, die als Teil der Datenbank gespeichert sind, geprüft. Damit wird gewährleistet, daß keine Modifikation oder Fortschreibung die Zuverlässigkeit der Datenbank gefährdet.

Ein leistungsfähiger Zentral-Server hilft System-Administratoren bei der Steuerung von Sicherheitsthemen, z.B. wie Applikationen einge-

setzt werden können, wieviele Anwender eine gegebene Applikation gleichzeitig benutzen können und wer welche Applikationen zu einer bestimmten Zeit nutzen darf.

Gemeinsame dezentrale Dienstleistungen bedeuten niedrigere Kosten pro Benutzer. Teure Hardware, wie 386- und 486-Computer, Coprozessor-Karten, numerische Coprozessor-Chips, Modems, Telefax-Karten, Mini/Großrechner-Kommunikationskarten etc., sowie teure Server-Software, können problemlos von einer Vielzahl von Teilnehmern gleichzeitig in einem Netz benutzt werden.

Indem die dezentrale Datenverarbeitung die Lebensdauer eines schwächeren 8086- und 80286-DOS-Arbeitsplatzrechners verlängert, erhöht sie dessen Rentabilität. Eine Client/Server-Architektur kann einen beachtlichen Teil der Funktionen von Arbeitsplatzrechnern übernehmen, mehr Speicherkapazität für Front-end-Applikationen zur Verfügung stellen und bietet so die Möglichkeit, an den Vorteilen der neuesten Netzwerktechnik-Generation zu partizipieren.

Die Client/Server-Datenverarbeitung führt zu einer erhöhten Wirtschaftlichkeit bei der Entwicklung von Applikationen sowie bei den gesamten System-Hardware- und Software-Kosten. Software-Entwickler brauchen keine eigenen Back-end-Dienste entwickeln. Sie können bestehende Dienstleistungen nutzen und sich auf andere, für die Anwender nützlichere Bereiche konzentrieren, wie z.B. Front-end-Verbesserungen und die Weiterentwicklung der Bedieneroberfläche.

APIs

Eine große Zahl von APIs macht die Client/ Server-Datenverarbeitung möglich. Nachstehend sind die wichtigsten sieben API-Kategorien für vernetzte Applikationen, die für LAN-Manager und LAN-Server einheitlich sind, aufgeführt. Die restlichen gemeinsamen APIs werden für Netzwerk-Utilities und Administration verwendet.

»Named Pipes«-APIs

LAN-Manager und LAN-Server erweitern die Möglichkeiten der MS OS/2-Pipes oder der Interprozeß-Kommunikation auf das gesamte Netz. Diese Erweiterung oder Umleitung der MS OS/2-Interprozeß-Kommunikation auf das gesamte Netzwerk wird »Named Pipes« genannt.

Named Pipes sind Zweiwege-Interprozeß-Kommunikationskanäle, über die an jedem Punkt des Netzwerks gesendet und empfangen werden kann. Das Lesen und Schreiben einer Pipe ähnelt sehr stark dem Lesen und Schreiben einer Datei. Named Pipes sind die häufigste Methode zum Austausch von Informationen zwischen Front-end-PCs und Back-end-Server-Applikationen.

Programme können die gleiche Named pipe-Schnittstelle für die Kommunikation zwischen Prozessen, die auf derselben Maschine laufen, oder für zwei Prozesse, die auf getrennten Maschinen im Netz laufen, benutzen. Daraus folgt, daß jede Applikation, die aus zwei über Named Pipes kommunizierenden Prozessen besteht, eigentlich eine Netzwerk-Applikation ist. Um beide Prozeßfunktionen gemeinsam in einem Netz zu betreiben, sind keine Änderungen erforderlich.

Named Pipes sind einfach in der Handhabung und bieten eine exzellente Fehlerbehebung. Sie wurden speziell unter dem Gesichtspunkt entwickelt, eine abgesetzte Netzwerk-Interprozeß-Kommunikation zu bieten, die von Software-Entwicklern mit wenig oder keiner Netzwerk-Applikationserfahrung implementiert werden kann. Außerdem ist es erheblich einfacher, für Named Pipes Software zu schreiben als für NetBIOS. NetBIOS ist ein Protokoll auf niedrigerem Niveau und erfordert vom Programmierer, daß er sich mit Dingen wie der Öffnung und Schließung von Netzwerk-Verbindungen, Netzwerk-Fehlerbedingungen usw. beschäftigt. Der Aufrufmechanismus der Named Pipes ist einfach und doch leistungsfähig. Eine einzige Named pipe-Funktion übernimmt die Funktionen vieler NetBIOS-Aufrufe.

Mailslot-APIs
Bestimmte Arten von Nachrichten, die im Netz zwischen Prozessen ausgetauscht werden, erfordern keinen vollständigen Duplex-Übertragungskanal. Ein benannter Mailslot ist unter diesen Bedingungen das optimale Mittel für die Handhabung von Meldungen.

Ein Mailslot ist ein Einweg-Interprozeß-Kommunikationsmechanismus. Im Prinzip ist er genau, was der Name vermuten läßt: Ein Platz, an dem ein beliebiger Rechner in einem Netz eine Meldung hinterlegen kann. Sobald ein Arbeitsplatzrechner oder ein Server einen Mailslot eröffnet, kann jeder Prozeß, der auf verschiedenen Rechnern im Netz läuft, einfach mit Namensangabe an ihn senden. In diesem Sinn sind solche Mailslots »Viele senden an einen«-Kommunikationsmechanismen. Jeder Rechner, der den Namen eines Mailslots kennt, kann an den Mailslot schreiben, und nur der Prozeß, der den Mailslot eröffnet hat, kann den Inhalt lesen. Mailslots können jedoch ebenso als eine Form der »Viele senden an viele«-Kommunikation verwendet werden, indem einfach mehrere Rechner unter dem gleichen Namen Mailslots eröffnen.

Mailslots bieten viele interessante Anwendungen, wie »dynamisches Suchen« oder Lastverteilung. Beim »dynamischen Suchen« sucht ein Arbeitsplatzrechner eine besondere Netzwerk-Ressource, beispielsweise eine Server-gesteuerte Kalenderapplikation. Der Arbeitsplatzrechner muß nicht wissen, wo sich die Kalenderapplikation im Netz befindet, um eine Konversation zu beginnen. Er hinterlegt einfach eine Meldung in einem vereinbarten Mailslot. Im Lastverteilungs-Szenario können Kopien der gleichen Server-Kalenderapplikation auf mehreren Servern im Netz laufen. Jeder hat ein Mailslot unter dem gleichen Namen eröffnet. Wenn ein Client-Arbeitsplatzrechner versucht, eine Konversation mit einer Server-Kalenderapplikation zu beginnen und eine Meldung in einer Mailslot hinterlegt, bekommen alle Kalender-Server diese Meldung. Die verschiedenen Kalender-Server können dann entscheiden, wer tatsächlich die Anfrage des Anwenders bedient.

Service-APIs
Mit Hilfe der LAN-Manager/LAN-Server-Service-APIs können Server-Applikationen als Netzwerk-Dienste installiert werden.

Viele der LAN-Manager/LAN-Server-Komponenten sowie der Ashton-Tate/Microsoft SQL-Server und DCA/Microsoft Communications Server sind als Netzwerk-Dienste konfiguriert. Der wichtigste Vorteil bei der Installation der Server-Applikationen als Netzwerk-Dienste besteht darin, daß ein System-Administrator in der Lage ist, standardmäßige LAN-Manager/LAN-Server-Schnittstellen und Einrichtungen für das Starten, Stoppen, Unterbrechen und Fortfahren von Abläufen zu benutzen. Ebenso können Abläufe dynamisch im Netz positioniert werden.

Warn- und Meldungs-APIs
Über Warn- und Meldungs-APIs können Server-gesteuerte Applikationen Meldungen an die LAN-System-Administratoren wie auch an die Benutzer des Systems senden. Eine Server-Datenbank-Applikation könnte Meldungen, wie »unerlaubter Benutzerzugriff versucht«, »Datenbank-Fehler entdeckt« oder »Leistungsabfall – schlage Datenbank-Verdichtung vor«, an den System-Administrator richten, sobald eine entsprechende Situation entsteht.

Server-Prozesse können ebenso selbst auf Warnmeldungen achten. Es ist z.B. möglich, daß ein Server-Prozeß, der mit einem Notstromgerät versorgt wird, die Netzwerk-Warnung aussendet, daß der Strom unterbrochen wurde und das Netzwerk in wenigen Minuten zusammenbricht. Wenn andere Server-Prozesse diese Meldung erhalten, können Sie eine entsprechende Abschalt-Routine durchführen.

Server- und Share-APIs
Die Server-APIs unterstützen die Applikationen bei der Feststellung, welche Server mit dem Netz verbunden sind und welche Ressourcen auf jedem Server zur Verfügung stehen. Auf diese Weise können Arbeitsplatzrechner-Programme dem Benutzer eine Server-Auswahlliste zur Verfügung stellen, aus der er den Server wählt und ein Merken der Server-Namen überflüssig macht. Die Share-APIs ermöglichen es, Server-Programmen die Anzahl der gleichzeitigen Applikations-Benutzer einzuschränken. Hersteller von Applikationen können somit auf einfachere Weise Versionen des gleichen Produktes vermarkten, die unterschiedliche Benutzerzahlen unterstützen.

Zusammenfassung

Die dezentrale Datenverarbeitung ist die Zukunft des PC-Netzwerk-Betriebs. Neben Named pipe-APIs, Mailslot-APIs, Warn- und Meldungs-APIs, Service-APIs sowie Server- und Share-APIs bieten LAN-Manager und LAN-Server die erforderlichen Einrichtungen, um den Anforderungen der hochentwickelten Netzwerk-Applikationen gerecht zu werden. Software-Entwicklern wird eine hervorragende Basis geboten, auf der sie umfassende, dezentrale Applikationen entwickeln können. Einige dieser dezentralen Applikationen stehen bereits zur Verfügung, viele andere sind noch in der Entwicklung. Das Ergebnis wird eine außergewöhnliche Erhöhung der Netzwerk-Produktivität sein.

Gemeinsame API-Kategorien:

LAN-Manager von Microsoft und LAN-Server von IBM

Kategorie	Anwendung
Alert	Meldet an Netzwerk-Service-Programme und Applikationen Ereignisse auf dem Netzwerk.
Character Device	Steuerung gemeinsamer Ausgabe-Einheiten und der zugehörigen Queues.
Connections	Listet alle Verbindungen von einer Arbeitsstation im Netz zu einem Server oder alle Verbindungen über den Server zu einer gemeinsamen Ressource.
Files	Zeigen an, welche Dateien, Geräte und Pipe-Ressourcen auf einem Server geöffnet sind und schließen eine dieser Ressourcen, sofern es notwendig ist.
Mailslot	Einweg-Prozeß-zu-Prozeß-Kommunikation (IPC) über das Netzwerk.
Message	Senden, Empfangen, Lesen, Aufzeichnen und Weiterleiten von Nachrichten.
Pipes	Prozeß-zu-Prozeß-Kommunikation (IPC) über das Netzwerk mittels Named Pipes.
Remote Utilities	Kopieren und Verschieben von abgesetzten Dateien, abgesetzte Ausführung eines Programms und Zugriff zur Systemzeit-Information auf einem abgesetzten Server.
Server	Ermöglicht die ferngesteuerte Verwaltung von auf einem lokalen oder abgesetzten Server auszuführenden Tasks.
Service	Installation und Steuerung der Netzwerk-Service-Programme.
Sessions	Steuerung der Netzwerk-Sessions, die zwischen Arbeitsstationen und Servern eingerichtet sind.
Shares	Steuerung der gemeinsamen Ressourcen, wie Platten-Laufwerke, Drucker und Named Pipes.
Use	Prüfung oder Steuerung von Verbindungen (uses) zwischen Arbeitsstationen und Servern.
Workstation	Steuerung des Betriebs der Arbeitsstationen.

Die IBM- und Microsoft OS/2 LAN-Programmier-Plattform bietet 13 gemeinsame API-Kategorien (von insgesamt 83 APIs), die vom IBM-LAN-Server und vom Microsoft LAN-Manager unterstützt werden. Unabhängige Software-Hersteller können somit leistungsfähige dezentrale Applikationen erstellen, die sowohl auf den IBM OS/2 LAN-Produkten als auch unter dem Microsoft LAN-Manager ohne Modifikationen laufen, wenn sie für den gemeinsamen API-Satz geschrieben sind.

Alert-Kategorie

Die Funktionen in der Alert-Kategorie bilden ein System zur Meldung von Ereignissen auf dem Netzwerk an Netzwerk-Server-Programme und Applikationen.

API	Beschreibung
NetAlertRaise	Unterrichtet alle in der Alert-Tabelle eingetragenen Arbeitsstationen über das Stattfinden eines bestimmten Ereignisses.
NetAlertStart	Zeigt einer Arbeitsstation an, daß sie über ein bestimmtes Ereignis auf dem Netzwerk benachrichtigt wird.
NetAlertStop	Entfernt einen Arbeitsstations-Eintrag aus der Alert-Tabelle.

Character Device-Kategorie

Die Funktionen in der Character Device-Kategorie steuern gemeinsam genutzte alphanumerische Ausgabe-Einheiten und die zugehörigen Queues.

API	Beschreibung
NetCharDevControl	Schließt zwangsweise eine alphanumerische Ausgabe-Einheit.
NetCharDevEnum	Listet alle alphanumerischen Ausgabe-Einheiten in einer gemeinsamen Ausgabe-Einheiten-Queue des Servers.
NetCharDevGetInfo	Liefert Informationen über eine bestimmte alphanumerische Ausgabe-Einheit in einer gemeinsamen Device-Queue des Servers.
NetCharDevQEnum	Listet alle Queues für alphanumerische Ausgabe-Einheiten eines Servers.
NetCharDevQGet-Info	Liefert Informationen über eine bestimmte Ausgabe-Einheiten-Queue eines Servers.
NetCharDevQPurge	Löscht alle noch nicht bearbeiteten Anforderungen aus einer Queue für alphanumerische Ausgabe-Einheiten.
NetCharDevQPurge Sel	Löscht alle von einem bestimmten Computer gesendeten anhängigen Anforderungen, die in einer Queue für alphanumerische Ausgabe-Einheiten auf die Ausführung warten.
NetCharDevQSet-Info	Ändert den Status einer Queue für alphanumerische Ausgabe-Einheiten auf einem Server.

Connections-Kategorie

Die NetConnectionEnum-Funktion erzeugt eine Liste aller Verbindungen, die von einer Arbeitsstation zu einem Server erstellt wurden oder alle Verbindungen zu einer gemeinsam genutzten Ressource.

API	Beschreibung
NetConnection Enum	Listet alle Verbindungen zwischen Arbeitsstationen und Ressourcen auf einem Server oder alle Verbindungen, die innerhalb einer Session bestehen.

Files-Kategorie

Die Funktionen in der Files-Kategorie bilden ein System zur Überwachung, welche Datei, Einheit und Pipe-Ressourcen auf einem Server geöffnet sind. Falls notwendig, wird eine der Ressourcen geschlossen.

API	Beschreibung
NetFileClose	Schließt zwangsweise eine Ressource, falls ein Systemfehler ein reguläres Schließen über die DOS-Close-Funktion verhindert.
NetFileGetInfo	Liefert eine Information über die Öffnung einer Server-Ressource.

Mailslots-Kategorie

Die Funktionen in der Mailslots-Kategorie ermöglichen eine Einweg-Prozeß-zu-Prozeß-Kommunikation (IPC).

API	Beschreibung
DosDeleteMailslot	Löscht einen Mailslot und legt alle gelesenen oder ungelesenen Nachrichten ab.
DosMailslotInfo	Sendet Informationen über einen bestimmten Mailslot zurück.
DosMakeMailslot	Erzeugt einen Mailslot und bestätigt seine Nutzbarkeit.
DosPeekMailslot	Liest die nächste Nachricht in einem Mailslot ohne Daten zu entfernen.
DosReadMailslot	Liest und entfernt dann die zuletzt eingegangene Nachricht in einem Mailslot (prioritätsorientiert).
DosWriteMailslot	Schreibt eine Nachricht in einen bestimmten Mailslot.

Message-Kategorie

Die Funktionen in der Message-Kategorie dienen zum Senden, Empfangen, Lesen, Aufzeichnen und Weiterleiten von Nachrichten.

API	Beschreibung
NetMessageBuffer	Sendet die Informationen eines Puffers zu einem angemeldeten Teilnehmer oder einem bestimmten Computer.
NetMessageFileSend	Sendet eine Datei an einen registrierten Anwender oder einen bestimmten Computer.
NetMessageLog FileGet	Holt den Namen der Nachrichten-Log-Datei und den aktuellen Logging-Status (Ein oder Aus).
NetMessageLogFile Set	Spezifiziert eine Datei gemäß einer Log-Nachricht, die von einem registrierten Anwender kam und ermöglicht oder sperrt die Aufzeichnung.
NetMessageName Add	Registriert einen Anwender in der Message-Name-Tabelle.
NetMessageName Del	Löscht einen Teilnehmernamen aus der Message-Name-Tabelle.
NetMessageName Enum	Listet die Teilnehmernamen-Einträge in einer Message-Name-Tabelle auf.
NetMessageName Fwd	Ändert die Message-Name-Tabelle bei der Übermittlung einer Teilnehmer-Nachricht an einen anderen Teilnehmer.
NetMessageName GetIn	Holt Informationen über das Message Account eines Teilnehmers.
NetMessageName UnFwd	Stoppt die Übertragung einer Teilnehmer-Nachricht an einen anderen Teilnehmer.

Pipes-Kategorie

Die Funktionen in der Pipes-Kategorie steuern die Prozeß-zu-Prozeß-Kommunikation (IPC) über Named Pipes.

API	Beschreibung
DosBufReset	Löscht den Datenpuffer einer Named Pipe.
DosCallNmPipe	Öffnet eine Named Pipe, führt das Schreiben in eine Named Pipe aus, gefolgt durch das Lesen, und schließt dann die Pipe.
DosClose	Schließt eine Named Pipe.
DosConnectNmPipe	Wartet auf einen Client Process, um in diesem Falle eine Named Pipe zu öffnen.
DosDisconnect-NmPipe	Schließt zwangsweise eine Named Pipe, unterbindet einem Client Process jeden weiteren Zugriff darauf.

API	Beschreibung
DosMakeNmPipe	Erzeugt eine neue Named Pipe oder eine neue Instance einer existierenden Named Pipe und bestätigt die Bearbeitung.
DosOpen	Öffnet den Client Process am Ende einer Named Pipe und bestätigt die Bearbeitung.
DosPeekNmPipe	Liest die Daten in einer Named Pipe ohne sie zu entfernen.
DosQFHandState	Holt Informationen darüber, ob die Bearbeitung einer Named Pipe berechtigt ist und ob das Schreiben in abgesetzte Pipes erlaubt ist.
DosQHandType	Bestätigt den Typ einer bestimmten Bearbeitung.
DosQNmpHand-State	Bestätigt Informationen über den aktuellen Status einer Named Pipe.
DosQNmPipeInfo	Holt Informationen über die Größe der Eingangs- und Ausgangspuffer bestimmter Named Pipes sowie darüber, wieviele Instances verfügbar sind.
DosQNmPipeSem-State	Bestätigt Informationen über den Status eines Semaphores, der zu einer Named Pipe eines lokalen Computers gehört.
DosRead	Liest Daten aus einer Named Pipe.
DosReadAsync	Liest Daten asynchron aus einer Named Pipe und entfernt die Daten.
DosSetFHandState	Ändert den Open-Mode-Status einer Named Pipe.
DosSetNmpHand-State	Ändert den Lesemodus und Blocking-Mode-Status einer Named Pipe.
DosSetNmPipeSem	Ordnet ein Semaphore einem Client oder Server Process auf einer lokalen Named Pipe zu.
DosTransactNmPipe	Schreibt und liest eine Nachricht in/aus einer Named Pipe.
DosWaitNmPipe	Ermöglicht einem Client Process, auf eine verfügbare Instance einer Named Pipe zu warten.
DosWrite	Schreibt Daten in eine Datei oder Named Pipe.
DosWriteAsync	Schreibt Daten asynchron in eine Named Pipe.

Remote-Utilities-Kategorie

Die Funktionen in der Remote-Utilities-Kategorie ermöglichen es der Applikation, abgesetzte Dateien zu kopieren und zu übertragen, ein abgesetztes Programm zu verarbeiten und auf die Systemzeit-Information eines abgesetzten Servers zurückzugreifen.

API	Beschreibung
NetRemoteCopy	Kopiert eine oder mehrere Dateien von einer Stelle zu einer anderen auf einem abgesetzten Server.
NetRemoteExec	Arbeitet ein Programm ab, das auf einem abgesetzten Server gespeichert ist.
NetRemoteMove	Verschiebt eine oder mehrere Dateien von einer Stelle zu einer anderen auf einem abgesetzten Server.
NetRemoteTOD	Sendet die Systemzeit des Servers zurück.

Server-Kategorie

Die Funktionen in der Server-Kategorie ermöglichen die ferngesteuerte Verwaltung von Tasks, die auf einem lokalen oder abgesetzten Server ausgeführt werden.

API	Beschreibung
NetServerAdmin Comma	Führt einen Befehl auf dem Server aus.
NetServerDiskEnum	Holt eine Liste der lokalen Laufwerke eines Servers.
NetServerEnum	Liefert Informationen über alle erkennbaren Server innerhalb einer LAN-Gruppe (Domain in LAN-Server).
NetServerGetInfo	Holt Informationen über einen bestimmten Server in einer von drei möglichen Detail-Klassifizierungen.
NetServerSetInfo	Setzt die Betriebsparameter eines Servers.

Service-Kategorie

Die Funktionen in der Service-Kategorie installieren und steuern Netzwerk-Service-Programme.

API	Beschreibung
NetServiceControl	Steuert die Operationen des Netzwerk-Services.
NetServiceEnum	Holt Informationen über alle Netzwerk-Service-Programme, die auf einem Server installiert sind.
NetServiceGetInfo	Holt Informationen über ein bestimmtes installiertes Netzwerk-Service-Programm.
NetServiceInstall	Installiert ein Netzwerk-Service-Programm auf einem Server.
NetServiceStatus	Setzt Status- und Code-Informationen eines Netzwerk-Service-Programms.

Sessions-Kategorie

Die Funktionen in der Sessions-Kategorie steuern Netzwerk-Sessions, die zwischen Arbeitsstationen und Servern stattfinden.

API	Beschreibung
NetSessionDel	Beendet eine Session zwischen einer Arbeitsstation und einem Server.
NetSessionEnum	Liefert Informationen über alle aktuellen Sessions für einen Server.
NetSessionGetInfo	Holt Informationen über eine Session, die zwischen einer bestimmten Arbeitsstation und einem Server stattfindet.

Shares-Kategorie

Die Funktionen in der Shares-Kategorie steuern gemeinsame Ressourcen, wie Platten-Laufwerke, Drucker und Named Pipes.

API	Beschreibung
NetShareAdd	Erzeugt eine gemeinsam nutzbare Ressource auf einem Server.
NetShareCheck	Anfrage, ob ein Server eine gemeinsame Einheit nutzt.
NetShareDel	Entfernt einen Share-Namen aus der Server-Liste der gemeinsamen genutzten Ressourcen.
NetShareEnum	Holt Share-Informationen über jede gemeinsam genutzte Ressource eines Servers.
NetShareGetInfo	Holt Informationen über eine bestimmte gemeinsam genutzte Ressource auf einem Server.

API	Beschreibung
NetShareSetInfo	Setzt einen neuen Share-Parameter oder Parameter für eine gemeinsam genutzte Einheit.

Use-Kategorie

Die Funktionen der Use-Kategorie prüfen oder steuern Verbindungen (uses) zwischen Arbeitsstationen und Servern.

API	Beschreibung
NetUseAdd	Richtet eine Verbindung zwischen einem lokalen oder NULL-Einheiten-Namen und einer gemeinsam genutzten Einheit durch Neueintrag des lokalen oder NULL (UNC) Einheiten-Namen für die gemeinsam genutzte Ressource ein.
NetUseDel	Beendet eine Verbindung zwischen einem lokalen oder UNC-Einheiten-Namen und einer gemeinsamen Ressource.
NetUseEnum	Listet alle aktuellen Verbindungen zwischen der lokalen Arbeitsstation und Ressourcen eines abgesetzten Servers auf.
NetUseGetInfo	Holt Informationen über die Verbindung zwischen einer lokalen Einheit und einer gemeinsamen Ressource.

Workstations-Kategorie

Die Funktionen in der Workstations-Kategorie steuern die Operation von Arbeitsstationen.

API	Beschreibung
NetWkstaGetInfo	Sendet Informationen über die Konfigurations-Komponenten einer Arbeitsstation zurück.
NetWkstaSetInfo	Konfiguriert eine Arbeitsstation.
NetWkstaSetUID	Diese Funktion logt gleichzeitig einen User-Namen auf das Netzwerk, ändert das Paßwort eines User-Namens und logt einen User-Namen aus dem Netzwerk aus.

Microsoft liefert Version 2.11 von Windows/286 und Windows/386 aus

Die neue Windows Version 2.11, die gegenüber der Vorversion 2.1 hinsichtlich ihrer Speicherverwaltung grundlegend überarbeitet wurde, ist ab sofort verfügbar. Insbesondere hat Microsoft die Verwaltung des EMS-Speichers (Expanded Memory Specification) reorganisiert, der über Bank Switching den indirekten Zugriff auf den Arbeitsspeicherbereich jenseits der 640 Kbyte-Barriere von MS-DOS ermöglicht. So wurde zum Beispiel die Schaltgrenze zwischen Small und Large Frame EMS im Kernel von 208 Kbyte auf 240 Kbyte erhöht. Für den Anwender hat dies den Vorteil, daß großen Windows-Applikationen wesentlich mehr Speicher zur Verfügung steht. Schwierigkeiten, die zum Beispiel bei der Arbeit mit Microsoft Excel unter Microsoft Windows 2.1 aufgetreten sind, werden mit der Version 2.11 beseitigt.

Microsoft hat aber auch an anderen Stellen das Leistungsverhalten von Windows grundlegend verbessert. So erkennt die EMS-Speicherverwaltung von Windows 2.11 einen installierten HIMEM.SYS-Treiber und kann dem System dadurch zusätzlichen Speicher zur Verfügung stellen. Der HIMEM.SYS-Treiber erlaubt es Applikationen, 64 Kbyte Speicherraum jenseits der 1 Mbyte-Speichergrenze wie gewöhnlichen Arbeitsspeicher zu adressieren. Nach Abzug für Systemaufgaben bleibt daher auf Maschinen mit mehr als 1 Mbyte Speicher für Anwendungen ein direkt adressierbarer Arbeitsspeicher von rund 684 Kbyte statt 640 Kbyte. Das bietet nicht nur großen Applikationen mehr Spielraum, sondern wirkt sich auch positiv auf die Performance des Gesamtsystems aus. Beispielsweise wurde die Systemleistung im Druckverhalten verbessert: Um das Drucken zu beschleunigen, ist die Geschwindigkeit des Druckerspoolers erhöht worden.

Neben diesen grundlegenden Änderungen wurden viele weitere Ergänzungen und Verbesserungen vorgenommen: Für Windows/386 steht jetzt beispielsweise auch ein 8514-Treiber für hochauflösende IBM PS/2-Bildschirme zur Verfügung. Des weiteren wurden folgende Druckertreiber überarbeitet: Com-Treiber, Postscript-Treiber, AppleTalk-Library, HPPCL-Treiber, Toshiba-24-Nadeldrucker-Treiber und Tastaturtreiber.

Das Update von Microsoft Windows/286 2.1 auf die neue Version Windows/286 2.11 bzw. von Windows/386 2.1 auf Windows/386 2.11 ist kostenlos. Ab 1. Oktober 1989 gelten alle anderen Preise (inkl. MwSt., unverb. Preisempf.) wie folgt:

Windows/286:

Neupreis	DM 547,--
Update von Version 1.03	DM 285,--
Update von Version 2.03	DM 57,--

Windows/386:

Neupreis	DM 855,--
Update von Version 1.03	DM 455,--
Update von Version 2.03	DM 57,--

Windows Toolkit:

Neupreis	DM 1.425,--
Update von Version 1.03	DM 547,--
Update von Version 2.03	DM 57,--

Microsoft QuickC 2.0 jetzt auch mit QuickAssembler

Microsoft QuickC mit QuickAssembler ist eine leistungsstarke Ergänzung der Quick-Sprachenfamilie, die einen in die QuickC-Umgebung integrierten vollständigen Macro Assembler bietet. Damit ist es jetzt leichter als je zuvor, C, gemischtes C, Assemblersprache und sogar Stand-Alone-Assembler-Programme bzw. -Routinen zu schreiben.

Der C-Teil des QuickC Compilers/QuickAssemblers ist identisch mit der bewährten QuickC 2.0 Umgebung. Der Assembler-Teil umfaßt einen integrierten Macro Assembler. Das Paket ist ab sofort im Handel erhältlich.

Die Compiler- und Assemblerbereiche des Produktes arbeiten folgendermaßen zusammen: Falls die aktuelle Datei eine ASM-Erweiterung hat, wird der Assembler durch den Befehl MAKE.EXE aufgerufen. Sollte die Datei eine C-Erweiterung aufweisen, wird der C-Compiler aufgerufen. Bei Verwendung einer Make-Datei, die sowohl C- als auch ASM-Module enthält, ruft die Umgebung automatisch den C-Compiler und gegebenenfalls den Assembler auf.

Der QuickAssembler wurde in die QuickC-Umgebung integriert, weil Microsoft QuickC 2.0 leistungsstarke Editing-, Debugging- und Compiling-Tools beinhaltet. Marktuntersuchungen zeigen, daß Assembler nach C die von C-Programmierern am häufigsten benutzte Sprache ist.

Durch den QuickAssembler ergeben sich einige Änderungen im Entwicklungsprozeß. Die Änderung hängt ab von der Art des Programms, das entwickelt wird. Falls der Programmierer Assembler-Programme schreibt, ist Microsoft QuickC/QuickAssembler die erste verfügbare integrierte Entwicklungsumgebung. Programmierer, die gemischte C- und Assembler-Programme schreiben, haben jetzt eine integrierte Entwicklungsumgebung, die mehrere Sprachen unterstützt. In der Vergangenheit war die Entwicklung von gemischtsprachlichen Programmen zeitraubend. Ein tyischer Ablauf hat vielleicht so ausgesehen:

1. Editieren Sie C-Code.
2. Verlassen Sie den Editor.
3. Compilieren Sie C-Code (zur Erstellung einer OBJ-Datei).
4. Gehen Sie ins MS-DOS.
5. Gehen Sie in den Editor und schreiben Sie die Assembler-Routine.
6. Verlassen Sie den Editor.
7. Assemblieren Sie die Routine (MS-DOS Befehlszeilen-Assembler).
8. Linken Sie C-OBJ- und Assembler-OBJ, um EXE zu erstellen.
9. Starten Sie CodeView und debuggen Sie (Abhängig davon, welche Art von Fehlern gefunden wird, entweder zu Schritt 1.

oder 3. zurückgehen; vorher in MS-DOS zurückkehren. Diese Schritte wiederholen, bis das Programm ordnungsgemäß arbeitet).

Bei QuickC/QuickAssembler könnte ein typischer Ablauf so aussehen:

1. Starten Sie QuickC/QuickAssembler und editieren Sie C-Code und Assemblerroutinen. (Anmerkung: Alle Informationen über C und Assembler sind über den Online Quick-Ratgeber verfügbar)
2. Betätigen Sie [F5] (Programm erstellen) und die Umgebung kompiliert, assembliert und linkt das Programm automatisch. In einer Sekunde wird ein EXE-Programm erstellt.
3. Beginnen Sie mit dem Debuggen. Da der Debugger in den Editor integriert ist, muß der Betriebsmodus nicht gewechselt werden. Gehen Sie – falls notwendig – zurück zu 1, ohne die Umgebung zu verlassen.

Microsoft QuickC 2.0 war die erste integrierte Entwicklungsumgebung mit einem eingebauten Inline-Assembler. Es ist also kein zusätzlicher, separater Assembler erforderlich, um auf den Inline-Assembler zuzugreifen. Der QuickC 2.0 Inline-Assembler ist in der Lage, Assembler-Code zeilenweise in den C-Code einzufügen. Dieser Prozeß ist jedoch auf den Inline-Code beschränkt, er gestattet keine Assembler-Routinen, Makros oder Datenspezifikationen.

Mit QuickC/QuickAssembler verfügt der Programmierer über die Leistungsfähigkeit des Microsoft Macro Assemblers 5.1. Darin beinhaltet sind:

- Leistungsstarke Datenspezifikationen einschließlich Datensätze und Strukturen.
- Mit dem von Microsoft entwickelten Interlanguage-Prozeß werden Makros und Anweisungen bearbeitet, die das Aufrufen anderer Microsoft Sprachen extrem einfach machen.
- Mit dem Inline-Assembler können keine Bibliotheken von Assemblerroutinen erstellt werden, auf die andere Programme Zugriff hätten. QuickC/QuickAssembler bietet die Standard-MASM-Anwendungseigenschaften.

QuickC/QuickAssembler unterscheidet sich vom Microsoft Macro Assembler in etlichen Punkten:

- QuickC/QuickAssembler ist eine Erweiterung der leistungsstarken QuickC Entwicklungsumgebung. Es ist kein Stand-Alone Microsoft Macro Assembler.
- QuickC/QuickAssembler ist nur auf MS-DOS lauffähig. MASM hingegen unterstützt sowohl MS-DOS als auch Microsoft OS/2.
- QuickC/QuickAssembler ist eine Real Mode-Applikation für 80286/80386-Prozessoren. MASM unterstützt Protected Mode und die Generierung von 80386 OP-Codes.
- Mit dem MASM können Programmierer Assembler-Routinen für beliebige andere Microsoft Sprachen schreiben, einschließlich C 5.1, QuickBasic 4.5, Basic 6.0, Pascal 4.0, FORTRAN 5.0 und COBOL 3.0. Die im MASM enthaltene Dokumentation und der integrierte CodeView-Debugger unterstützen ebenfalls alle vorstehenden Sprachen. QuickC/QuickAssembler kombiniert C mit Assemblersprache.

Microsoft QuickC 2.0 ist natürlich weiterhin auch ohne QuickAssembler erhältlich.

Starkes Software-Duo für Geschäftsgrafiken

Wer Grafiken schnell, komfortabel und professionell auf dem PC verarbeiten möchte, sollte jetzt zu einem Angebot greifen, das Microsoft befristet bis Ende Dezember macht: Zum »Paketpreis« von DM 1.311,-- (inkl. MwSt., unverb. Preisempf.) bietet das Softwarehaus das leistungsfähige Geschäftsgrafikprogramm Microsoft Chart Version 3.0 plus das eben von Ernst Tiemeyer erschienene Trainingsbuch »Geschäftsgrafiken mit Microsoft Chart 3.0 auf dem PC«.

Mit diesem »starken« Software-Duo können Anwender, die Zahlenwerte und Texte mit dem PC darstellen möchten, nicht nur einen raschen Einstieg in die Grafikverarbeitung schaffen, sondern sich auch anhand von problembezogenen Beispielen mit den grundlegenden und vielfältigen Gestaltungsmöglichkeiten von Microsoft Chart 3.0 vertraut machen. Die Lernfähigkeit wird dabei sowohl durch das in Microsoft Chart 3.0 integrierte Lernprogramm als auch durch das gute didaktische Konzept des Lehr- und Übungsbuches hervorragend unterstützt.

Deutsche Versionen von zahlreichen Microsoft-Macintosh-Programmen jetzt verfügbar

Microsoft Works 2.0 für den Macintosh

In Microsoft Works 2.0 für den Macintosh sind die fünf Funktionsbereiche Textverarbeitung, Tabellenkalkulation, Datenbank, Kommunikation und Grafik in einem einzigen Programmpaket vereint.

Dieses integrierte Paket ist die ideale Komplettausstattung für kompakte Macintosh-Systeme aber auch eine Ergänzung, die eine breite Grundfunktionalität allen »Spezialisten« gewährleistet.

Das deutsche Microsoft Works, das ab sofort im Handel verfügbar ist, bietet in der neuen Version 2.0 eine Reihe zusätzlicher Funktionen. So wurden die grafischen Fähigkeiten erweitert und verbessert. Ferner bieten Makros eine Automatisierung sich stets wiederholender Aufgaben.

Für die Erstellung von Grafiken enthält Microsoft Works 2.0 zahlreiche Werkzeuge zum

Zeichnen von Linien, Kreisen, Rechtecken, Winkeln und Polygonen sowie zum freihändigen Zeichnen. Diese grafischen Fähigkeiten können sowohl in der Tabellenkalkulation als auch in der Textverarbeitung angewendet werden.

Drei weitere Verbesserungen stehen in allen fünf Programmen des Pakets zur Verfügung: Farbunterstützung für den Macintosh II, Seitenansicht am Bildschirm (zur Begutachtung einer Seite vor dem Ausdruck) sowie ein Makro-Rekorder, der zur Aufzeichnung eines Arbeitsablaufs für die spätere wiederholte Ausführung dient.

Die Optimierung der Serienbrieffunktionen erlaubt die Verwendung mehrspaltiger Adreßaufkleber. Das Tabellenkalkulations-Modul bietet nun 256 Spalten und bis zu 16.382 Zeilen. Das Datenbank-Modul wurde durch Einfügen von Datum-/Zeit-Funktionen und durch flexiblere Auswertungs-Fähigkeiten verbessert. Datenimport und -export verschaffen Microsoft Works 2.0 auf dem Macintosh Kompatibilität zu Microsoft Word und Microsoft Excel durch Dateien im RTF- sowie SYLK-Format.

Microsoft Works 2.0 für den Macintosh ist ab sofort zu einem Preis von DM 980,-- (inkl. MwSt.; unverbindl. Preisempf.) im Fachhandel erhältlich. Besitzer einer früheren Works-Version können für DM 279,-- (inkl. MwSt.; unverbindl. Preisempf.) auf Microsoft Works 2.0 umsteigen.

Microsoft Word 4.0 für den Macintosh

Ab sofort verfügbar ist die deutsche Version von Microsoft Word 4.0 für den Macintosh, in dessen Entwicklung Microsoft die Anregungen von weltweit über 400.000 Microsoft Word-Anwendern einfließen ließ, um das erfolgreichste Macintosh-Programm entsprechend zu optimieren. Microsoft Word 4.0 für den Macintosh weist gegenüber der Vorversion daher wichtige Weiterentwicklungen auf, z.B.:

1. Schnelles Arbeiten durch individuelle Anpassung
Die Menüs in Word 4.0 können komplett den persönlichen Ansprüchen angepaßt und entsprechend gestaltet werden.
2. Optimale Darstellung von Dokumenten
Um Dokumente optimal darzustellen, bietet Word 4.0 jetzt vier unterschiedliche Bearbeitungsansichten: Gliederungsansicht, Fahnenansicht, Seitenansicht und Druckansicht zum Editieren im WYSIWYG-Modus.
3. Einsatz neuer Tabellenfunktionen
Mit den neuen Tabellenfunktionen in Microsoft Word 4.0 kann die Erstellung u.a. von Preislisten, Tabellen oder anderen spaltenartigen Übersichten schneller und einfacher als bisher gestaltet werden.
4. Absolutes Positionieren
Mit dem neuen Word 4.0 lassen sich auch DTP-Aufgaben erledigen. So erlaubt z. B. die Möglichkeit der absoluten Positionierung von Objekten, eine Grafik fest an einer Stelle der Seite zu fixieren. Der Text fließt dann ein- oder mehrspaltig um das Objekt herum.

Da die meisten Macintosh Anwender mehrere Softwarepakete einsetzen, hat Microsoft vor allem auch auf die Integrationsfähigkeit der neuen Word-Version besonderen Wert gelegt. Die dynamische Verknüpfung von Microsoft Word 4.0 mit anderen Software-Programmen macht es einfacher denn je, beispielsweise Geschäftsberichte, die Text, Zahlen und Grafiken verbinden, zu erstellen.

Microsoft Word 4.0 für den Macintosh setzt als Systembasis einen Macintosh Computer mit 512 Kbyte RAM und zwei 800-Kbyte-Floppydisk-Laufwerken oder einer Festplatte voraus. Das Programm ist kompatibel mit AppleShare und MultiFinder.

Microsoft Word 4.0 für den Macintosh ist zu einem Preis von DM 1.585,-- (inkl. MwSt.; unverb. Preisempf.) im Fachhandel erhältlich. Besitzer einer früheren Version können für DM 279,-- (inkl. MwSt.; unverb. Preisempf.) auf die neue Version umsteigen.

Deutsche Version von Microsoft QuickBasic für den Macintosh

Mit der deutschen Version von Microsoft QuickBasic 1.0 steht Macintosh Anwendern ab sofort dieses vielseitige und leistungsfähige Entwicklungswerkzeug zur Verfügung, in dem die Vorteile eines Basic Interpreters mit denen eines Basic Compilers kombiniert wurden.

Die Interpreter-Vorteile liegen in der Programmerstellung mit Editor und Debugger. Die Vorteile des Compilers machen sich hingegen in der Schnelligkeit der Programmausführung bemerkbar. In QuickBasic geschriebene und kompilierte Programme sind eigenständige Applikationen, die nach ihrer Fertigstellung weder Interpreter noch Compiler zum Ablauf benötigen.

Microsoft QuickBasic eröffnet damit für den Macintosh neue Dimensionen. Dabei ist die einfach beherrschbare Sprache Basic eine interessante Alternative in der Programmierung. Selbst komplexe Aufgaben sind unter der neuen, verbesserten Bedienungsoberfläche, die der vom Macintosh gewohnten Umgangsweise entgegenkommt, leicht lösbar. Zu den besonderen QuickBasic Eigenschaften gehören die Tonerzeugungs- und Farbgrafik-Unterstützung. Hinzu kommt, daß QuickBasic die Fähigkeiten der Prozessoren 68020, 68881 und höher nutzt.

Durch die Kombination von Compiler und Interpreter auf einer gemeinsamen Bedieneroberfläche lassen sich Kompilierung und Ausführung eines Programms auf einen einfachen »Klick« mit der Maus reduzieren. Der Zugriff auf die Macintosh Toolbox, eine Subroutinen-Bibliothek, gewährleistet ferner, daß Anwendungsprogramme auf einfache Weise mit Macintosh Standard-Interfaces versehen werden und gewohntes Aussehen sowie bekannte Bedienungsvorteile erhalten.

Die Eigenschaften des Quellcode-Debuggers sind geprägt durch Einzelschritt-Abarbeitung, Trace-Funktion und die Möglichkeit, Unterbrechungen zu setzen. Im Direkt-Modus können nach jeder Programmunterbrechung die Inhalte von Variablen angezeigt sowie Kommandos unabhängig vom Programm ausgeführt werden. Die jederzeit verfügbare Online-Hilfe von QuickBasic informiert auf Knopfdruck über Basic-Vokabeln, Bibliotheksfunktionen und Aufrufe, die zu ROM-Routinen gehören.

Zum Lieferumfang gehören zahlreiche Programm-Beispiele, die den Einstieg in Basic erleichtern und viele der Microsoft QuickBasic-Besonderheiten anschaulich nahebringen. QuickBasic erfordert in dieser modernsten Version 1 Mbyte RAM im Macintosh Plus, SE oder II und mindestens ein doppelseitiges 800-Kbyte-Laufwerk. Die Verwendung einer Festplatte oder ein zweites 800-Kbyte-Laufwerk ist empfehlenswert.

Microsoft QuickBasic 1.0 ist in der deutschen Version ab sofort zu einem Preis von DM 399,- (inkl. MwSt.; unverb. Preisempf.) im Fachhandel erhältlich. Besitzer eines Microsoft Basic Interpreters und Basic Compilers können für DM 148,-- (inkl. MwSt.; unverb. Preisempf.) auf QuickBasic 1.0 umsteigen.

Apple-Testfahrt mit Microsoft Produkten

Apple Computer, Claris und Microsoft bieten ab sofort bis 31. Oktober dieses Jahres eine gemeinsame Testaktion an, bei der sich Anwender über die Leistungen der Macintosh Hard- und Software in der Praxis informieren können.

Für die Testaktion wird von Apple wahlweise ein Macintosh SE 2/20 oder SE/20 2/40 mit ISO-Tastatur zur Verfügung gestellt. Jeder Rechner beinhaltet sechs Softwareprodukte in einer Demo-Version, u.a.

- Microsoft Word 4.0 für den Macintosh
- Microsoft Excel 1.5 für den Macintosh
- Microsoft PowerPoint 2.0 für den Macintosh.

Damit bietet Microsoft Apple-Aspiranten einen praktischen Einblick in die Arbeit mit der im Macintosh-Markt führenden Standardsoftware für Textverarbeitung, Zahlenmanagement und Präsentationssoftware.

Wer an der gemeinsamen Testaktion teilnehmen möchte, hinterlegt ganz einfach bei einem Apple-Händler eine Nutzungsgebühr in Höhe von DM 100,– zuzüglich einer Kaution und erhält dafür die Testkonfiguration seiner Wahl sechs Wochentage zum Praxistest. Entscheidet sich der Anwender zum Kauf, erhält er als Geschenk ein dreimonatiges Abonnement wahlweise der »MACup« oder des »Macintosh Magazins«.

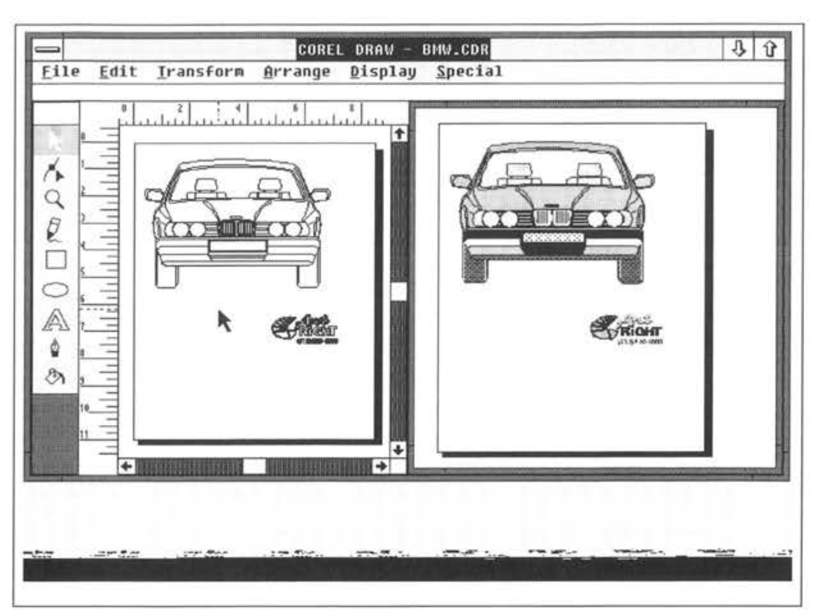
Microsoft System-center macht integrierte Bürokommunikation transparent

Mit der Eröffnung des weltweit ersten »System-Centers« in ihrer Düsseldorfer Niederlassung bietet Microsoft Großkunden und OEMs jetzt einen praxisnahen Blick in das Zeitalter der integrierten Bürokommunikation. Der führende Anbieter von Programmiersprachen, Anwendungsprogrammen und Betriebssystemen zeigt in diesem Demo-Zentrum, das unter Beteiligung führender Hard- und Software-Hersteller realisiert wurde, wie unter dem neuen Betriebssystemstandard MS OS/2 in lokalen Netzwerken die Zukunft der Büroarbeit aussehen wird – eine Zukunft, die mehr Effizienz am PC-Arbeitsplatz oder besser: in der PC-Arbeitsgruppe bietet.

Das neue Microsoft-System-Center bildet ein Forum zur Information über die aktuellen Komponenten der Systemsoftware, den neuesten Stand bei Applikationen sowie über die Funktionsweisen und Aufgaben eines lokalen Netzwerks (LAN). Die Grundlage bilden Rechner unterschiedlicher Hersteller, die über IBM Token-Ring-Adapter miteinander verbunden sind und unter den Betriebssystemen MS-DOS mit der grafischen Benutzeroberfläche Microsoft Windows und MS OS/2 mit Microsoft Presentation Manager laufen. Installiert wurden ferner der Microsoft LAN-Manager und der SQL-Server. Ergänzt wird die MS-DOS und MS OS/2 Systemsoftware durch verschiedene Applikationen der traditionellen Bereiche Textverarbeitung, Tabellenkalkulation und Grafik. Sie zeigen nicht nur die Einsatzmöglichkeiten von Standardapplikationen innerhalb eines Netzwerks, sondern dokumentieren auch die Leistungen der neuesten Microsoft Software, wie z.B. Microsoft Word 5.0, Microsoft Excel 2.1 und Microsoft Multiplan 4.0.

Gleichzeitig gibt Microsoft Großkunden und OEMs in dem neuen System-Center auch einen Einblick in Administration, Einrichtung und die Sicherheitsmechanismen von Netzwerken.

Christian Wedell, Geschäftsführer der Microsoft GmbH: »Das Microsoft-System-Center ist nicht nur Ausdruck einer rasanten Entwicklung in der Durchsetzung von neuen Standard-Betriebssystemen wie Microsoft OS/2. Es wird auch ständig weiter ausgebaut und demonstriert somit in der Praxis den jeweils aktuellsten Stand der Softwaretechnologie und gibt EDV-Perspektiven.« So werden z.B. ein Apple Macintosh und ein Unix-Rechner mit dem LAN Manager/X in das bestehende Netz eingebunden. Die Verbindung von PCs und Großrechnern kann in naher Zukunft durch die Installation des Microsoft Communications Server gezeigt werden. Auch die Demonstration von Multiprotocolling ist in Planung. Dabei unterstützt ein Server gleichzeitig mehrere Netzwerk-Architekturen.



Corel Draw mit neuen Schriftfunktionen und Möglichkeiten

Neu im Vertrieb von EDTZ ist die Version 1.1 von Corel Draw. Die neue Version von Corels Windows-Grafikprogramm umfaßt 45 Outline-Schriften, einen Schriftkonverter, zusätzliche Clip-Art-Dateien, Unterstützung der MS-Windows-Zwischenablage (Clipboard), den Import und Export von CGM-Dateien (Computer Graphics Metafile) sowie SCODL-Export zur Herstellung qualitativ hochwertiger Farbdias auf Dia- und Film-Recordern.

Schriftenkonverter unterstützt über 3000 Fonts

Corel Draw 1.1 wird mit einem WFNBOSS (sprich: Wuf-in-Boss) genannten Schriftenkonverter geliefert. Diese von Corel Systems, Ottawa, entwickelte Windows-Utility gibt dem Anwender Zugang zu Schriftenbildern anderer Hersteller und ermöglicht dem Grafiker die Benutzung von Schriften, die nicht in Corel Draw enthalten sind. WFNBOSS konvertiert andere Schriftenformate in echte On-Screen WYSIWYG-Schriften und erlaubt somit direktes Unterschneiden am Bildschirm (Kerning), interaktive Veränderung von Wort- und Buchstabenabstand, Bearbeitung von Buchstabenformen, Spezialeffekte wie Farbabläufe, Anpassung von Text an Kurven (Rundsatz etc.) und Ausgabe auf sämtlichen von Windows unterstützten Ausgabegeräten, einschließlich HP LaserJet HP PaintJet, Matrixdruckern, Plottern, PostScript-Druckern oder Matrix-Film-Recordern.

Zur Zeit unterstützt Corel Draw acht verschiedene Schriftenformate. Die bedeutendsten hiervon sind die Schriften von Adobe Systems und die Bitstream Outline-Schriften.

Anwender, die bereits Adobe-Fonts gekauft haben, können diese Investition jetzt mit Corel Draw 1.1 noch besser nutzen – die Schriften können direkt eingesetzt werden. Anwender von Corel Draw können sämtliche Adobe-Schriften direkt am Bildschirm manipulieren.

Die populäre Bitstream Fontware-Schriftenbibliothek wird mit Corel Draws WFNBOSS ebenfalls zugänglich. Außerdem unterstützt WFNBOSS die Digi-Fonts-Bibliothek (über 250 Outline-Schriften). Der Schrifteneditor Altsys Fontographer für den Macintosh ist eines von zwei Programmen zur Gestaltung von Schriften, das ebenfalls von Corel Draw unterstützt wird. Bei dem anderen handelt es sich um Z-Softs Publisher's Type Foundry, einen PC-gestützten Editor. Image Club, Casady & Greene, sowie Treacyfaces werden über das Fontographer-Format unterstützt. Darüber hinaus ist die Agfa Compugraphic-Schriftenbibliothek mit über 1200 Schriften in Corel Draw zugänglich. Außerdem unterstützt Corel Draw auch den HP Compugraphic Type Director und die URW Schriftenbibliothek (The Font Company).

Besitzer von nicht postcriptfähigen Druckern (z.B. dem HP LaserJet und dem HP PaintJet sind jetzt in der Lage, Adobe und andere Schriften auf diesen Geräten zu verwenden. Sämtliche leistungsstarken Funktionen von Corel Draw zur Schriftenmanipulation sind voll zugänglich.

Der Zugriff auf eine solche Vielzahl professioneller Schriften gibt Corel Draw eine einzigartige Position in der Welt des PC-gestützten Desktop Publishings.

Schriften

Die neue Version von Corel Draw beinhaltet 45 zusätzliche Corel Outline-Schriften aus 14 verschiedenen Schriftenfamilien. Das Programm verfügt somit über 102 Schriftschnitte im Lieferumfang. Sämtliche Corel Draw Schriften sind genaue Entsprechungen häufig verwendeter Schriften und enthalten vollständige internationale Zeichensätze. Der Wert dieser Schriften, die kostenlos in Carol Draw mitgeliefert werden, beträgt über DM 10.000,- (bei ca. DM 100,- je Schrift).

Alle Schriften können mit allen Corel Draw-Funktionen bearbeitet und auf sämtlichen Windows-Ausgabegeräten ausgedruckt werden.

Clip-Art Vorlagen – über 20.000 Bilder werden unterstützt

Co-Marketing-Beziehungen zwischen Corel und zwölf bedeutenden Clip-Art-Herstellern haben zu einem umfangreichen Angebot an Zeichnungsvorlagen für Corel Draw geführt. Die Version 1.1 enthält 300 Bilder; Anwender können aus einem Angebot weiterer 20.000 Bilder

wählen. Anstatt eigene Grafiker mit der Herstellung von Clip-Art-Vorlagen zu beschäftigen, bietet Corel Draw den Zugriff auf die Vorlagen-Bibliotheken von:

ArtRight, Casady & Greene, DreamMaker Software, Dynamic Graphics, Image Club, Marketing Graphics Inc., Metro Image Base, MicroMaps, Multi-Ad, New Vision, T-Maker und 3G.

Das Corel-Draw-Programmpaket enthält einen Clip-Art-Katalog, der eine ideale Basis für Zukäufe von Clip-Art-Vorlagen darstellt. Die unterstützten Grafikformate sind: TIFF, PCX, CGM, EPS sowie CDR.

MS-Windows Zwischenablage wird unterstützt

Corel Draw 1.1 unterstützt das MS-Windows-Clippboard. Anwender können Grafiken durch Ausschneiden und Einfügen zwischen Corel Draw und anderen Windows-Anwendungen austauschen. Das Corel Draw-Clipboard erlaubt den Import von Grafiken aus Excel, Arts&Letters, Micrografx, Pixie, PC-Paintbrush für Windows, HP Scanning Gallery, T/Maker Scrapbook+. Export ist möglich nach PageMaker, Ami und Xerox Presents. Von besonderer Bedeutung ist dabei die Kombination Excel und Corel Draw. Sie stellt eine Herausforderung an die »Lotus-1-2-3/Freelance«-Kombination dar. Excel-Charts können unter Corel Draw mit größeren und ausgefalleneren Schriften versehen werden oder mit Firmenlogos, Farbverläufen, farblich abgestuften Hintergründen und Clip-Art aufbereitet werden.

CGM Import/Export

Die Fähigkeit, CGM-Dateien zu importieren bzw. zu exportieren, verschafft Corel Draws Kompatibilität mit Industriestandards eine neue Dimension. Grafiken aus Harvard Graphics, Lotus Freeland Plus, Zenographics Mirage, Arts&Letters, Micrografx, ISSCO Displa, CAD und einer Vielzahl anderer Software für PCs, Minis und Mainframes sind jetzt zugänglich. Der Export von Dateien im CGM-Format macht Corel Draw 1.1 zu einem idealen Partner für Seitenlayoutprogramme (Ventura Publisher, PageMaker), Desktop Präsentation (Hotshot Presents, Xerox Presents) und Textverarbeitungsprogramme (MS-Word, WordPerfect 5.0, Ami).

Der Vorteil des CGM-Formats liegt darin, daß es sich um ein vektororientiertes Grafikformat handelt, das auflösungs- und geräteunabhängig ist und darüber hinaus Farbinformationen wiedergeben kann (CGM, Computer Graphics Metafile).

Herstellung von Farbdias via SCODL

Mit Hilfe der SCODL-Sprache kann Corel Draw Version 1.1 Farbdias erzeugen. Agfa-Matrix und Genegraphics sind zwei Hersteller, die SCODL unterstützen; diese Dia-Filmrekorder sind weltweit in Service-Büros im Einsatz. Mit Corel Draw können diese Filmrekorder jetzt qualitativ hochwertige Farbdias erzeugen, die von gestalterischen Möglichkeiten, der umfassenden Schriftenbibliothek und Clip-Art-Vorlagen Gebrauch machen.

In den USA ist Corel Draw inzwischen im Begriff, sich als der Industriestandard für PC-Grafikprogramme zu etablieren. Das weltweit vertriebene Programm erhielt unter anderem die Auszeichnung PC Magazine's Editor's Choice (27. Juni), PC World's Best Buy (Juli Ausgaben), und die 5-Sterne-Einstufung durch Publish! Magazine (ebenfalls Juli 89).

Info: EDTZ Hard- und Software GmbH, Friedrich-Ebert-Str. 16-18, 8012 Ottobrunn, Tel.: (089) 608700

Design/OA – Die Entwicklungsumgebung für visuelle Entwurfssysteme

Design/OA ist eine offene Architektur, die Funktionen zur Erzeugung von interaktiven, grafischen Anwendungsprogrammen und Front/Ends zur Verfügung stellt.

Design/OA basiert auf MS-Windows, X-Windows bzw. Macintosh Toolbox und bietet eine vollständige horizontale Programmierumgebung, die für alle drei Oberflächen identisch ist. Anwendungen können daher leicht zwischen diesen Systemen portiert werden. Mit Hilfe der zahlreichen Funktionen, die einen vollständigen Zugriff auf die Datenstrukturen der Diagramme und die Funktionen von MetaDesign ermöglichen, sind Anschlüsse an andere Werkzeuge, wie z.B. Data Dictionary oder Analyseprogramme, leicht implementierbar.

Anwendungsprogramme für Design/OA werden in der Sprache C geschrieben. Sie kontrollieren die Wechselwirkung zwischen dem Benutzer und dem Design-Kern. Der Systementwickler bestimmt die Art und Anzahl der Sorten von Objekten, die Regeln für die Verbindung von Objekten sowie die Semantik für die Beziehungen zwischen den Objekten. Design/OA stellt folgende Funktionsgruppen zur Verfügung:

- Funktionen zur Behandlung der Diagrammstruktur. Dazu gehören Funktionen zur Erzeugung von Objekten (Knoten, Verbindungen, Texten und Seiten) und zur Generierung von Listen (Knoten-, Verbindungs- und Seitenlisten).
- Funktionen zum Abfragen und zur Änderung von Objekteigenschaften. Dazu gehören Funktionen zum Lesen und Ändern der grafischen Attribute von Objekten wie beispielsweise das Einstellen des Linientyps, des Füllgebiets und der Schriftart.
- Funktionen zur Änderung der Bildschirmdarstellung. Dazu gehören Funktionen zur Ausrichtung von Knoten, zur Ausgabe von Statusmeldungen, zur Bestimmung des sichtbaren Seitenausschnitts und zur Änderung des Cursors (Fadenkreuz o.ä.).
- Funktionen, die dem Design-Kern die Existenz von Filterfunktionen im Anwendungsprogramm bekanntmachen. Sie

ersetzen die Standard-Funktionen des Design-Kerns.

• Funktionen zur Dialogbearbeitung. Dazu gehören Funktionen zum Aufrufen von Dialogfenstern und zur Abfrage von Benutzereingaben.

• Funktionen zur Behandlung der Menüs. Dazu gehören Funktionen zum Einrichten, Ändern, Verschieben und Löschen von Menüs und Menüeinträgen.

• Funktionen zur Erweiterung der Standard-Datenstruktur eines Objekts.

• Funktionen zum Aufruf der MetaDesign Standard-Menüs.

Für kleine Anwendungsprogramme steht auf dem Macintosh Design/Add-On zur Verfügung. Der Funktionsumfang von Design/Add-On ist identisch mit dem Funktionsumfang von Design/OA. Die Größe von Anwendungsprogrammen ist jedoch auf 32 Kbyte begrenzt. Design/Add-On ist damit eine kostengünstige Alternative zu Design/OA für die Entwicklung kleiner Anwendungsprogramme.
Info: C.I.T. GmbH, Postfach 514, 1000 Berlin 27, Tel.:(030) 4346561

Move'em – ein neuer Memory Manager

Die Albrecht Software Systeme GmbH kündigt nach »386MAX Professional« einen weiteren – neuen – Memory Manager der Qualitas Inc. an: »Move'em« – für PCs mit LIM 4.0-Hardware oder C&T CHIPSets.

Move'em lädt Gerätetreiber und residente Programme in freie Bereiche zwischen 640 Kbyte und 1 Mbyte. Die Möglichkeiten entsprechen dabei grundsätzlich dem bekannten »38MAX Professional«.

Der LIM 4.0-Standard wird zur Plattform für den Sprung über bekannte Restriktionen des DOS-Betriebssystems. Auch Rechner mit NEAT-Board bzw. C&T CHIPSets profitieren von Move'em. Für den Anwender bedeutet dies: Mit vielen der installierten PCs, XTs, ATs kann der DOS-Speicher entlastet werden – mehr Luft für DTP, CAD, Netzwerkanwendungen.

• Move'em konfiguriert den Rechner so, daß im Bereich unterhalb 1 Mbyte möglichst viel »DOS«-Speicher verfügbar wird. In Systemen, die auf der Grundplatine mit weniger als 640 Kbyte RAM ausgestattet sind, wird RAM-Speicher aus dem EMS-Bereich zum Auffüllen benutzt. Systeme mit Monochrom- oder CGA-Adapter können dabei bis zu 704 bzw. 736 Kbyte konventionellen DOS-Speicher erzielen.

• Move'em kann speicherresidente Programme und Treiber (wie z.B. auch Mouse, KEYBGR, DOS Print) oberhalb der 640-Kbyte-Grenze laden (in freie Bereiche zwischen 640 Kbyte bis 1 Mbyte). Dieser Bereich wird HIGH DOS Memory genannt. Wertvoller DOS-Speicher wird frei.

• Move'em ermittelt automatisch die optimale Anordnung der Treiber/Programme zur bestmöglichen Nutzung des Speichers.

• Move'em ist kompatibel zu Microsoft XMS (Extended Memory Specification) sowie UMB (Upper Memory Blocks) und bietet dem Memory Manager der XMS-fähigen Software eine zusätzliche Speichererweiterung.

• Move'em kann durch »Shadowing« der ROMs die Verarbeitung beschleunigen (Kopieren der ROM-Routinen in schnelles RAM).

Die Systemvoraussetzungen sind: MS-DOS oder IBM PC-DOS ab Version 3.0. Entweder ein 80286- oder 80386-Rechner mit einem NEAT-Board oder AT/386 Chipset von Chips & Technologies mit mindestens 1 Mbyte Gesamtspeicher oder ein 808x- bzw. 80286-Rechner (PC, XT, AT und Kompatible) mit EMS 4.0-Hardware (LIM 3.2-Hardware bzw. LIM 4.0-Softwaretreiber alleine genügt nicht den Anforderungen!) und mindestens 256 Kbyte auf der Erweiterungskarte.

Weiterhin aktuell für alle 386er und unverändert lieferbar ist der »386MAX Professional«. Dieser bietet zusätzlich eine EMS-Emulation nach LIM 4.0-Standard auf Basis von Extended Memory. 386MAX Professional kann grundsätzlich auf allen 386er Rechnern eingesetzt werden, die über mindestens 256 Kbyte Extended Memory verfügen. Im Gegensatz zu Move'em ist kein besonderer Chipsatz erforderlich.

386MAX Professional bleibt damit auch weiterhin das Tool für leistungsstarke Rechnertypen wie beispielsweise IBM PS/2 386er oder Compaq 386er.

Move'em wird 285,- DM kosten, 386MAX Professional kostet DM 333,-, (empf. Endverkaufspreise inkl. MWSt.).
Info: Albrecht Software Systeme GmbH, Mooswiesenstr. 11a, 8000 München 60, Tel.: (089) 882767

BKS-GRAPH, neuer umfangreicher Treiber-Support ab Version 2.6

BKS-GRAPH ist eine leistungsstarke C-Funktionssammlung auf der Basis der DIN-Norm GKS (Grafisches Kern-System). Es wird für die Betriebssysteme MS-DOS, OS/2, SCO XENIX, UNIX V und FlexOS angeboten. Mit der Version 2.6 wurde unter verschiedenen Betriebssystemen der Treiber-Support erweitert und verbessert. Die Version 2.6 wird zum November '89 freigegeben.

Alle Systeme beinhalten jetzt einen multifunktionalen Druckertreiber für Matrix- und Laser-Drucker. Der Treiber unterstützt Epson FX 80, LQ 5, LQ 850, IBM Grafik-Printer (60 dpi bis 240 dpi), NEC P2/P3/P6/P7/P6 Color, OKI 320/390/290 Color und HP Laserjet (75 dpi bis 300 dpi). Der HP-Plotter-Treiber ist um die Formate 7550 hoch und quer sowie 7475 hoch erweitert worden. Für SCO XENIX 386 ist ein neuer EGA/VGA-Kerneltrei-

ber verfügbar, der die Bildaufbau-Geschwindigkeiten wesentlich beschleunigt und vollen Mouse-Support beinhaltet. Außerdem wird für alle UNIX- und XENIX-Systeme ein TEK 4010/4014-Treiber und ein HP2393-Treiber ausgeliefert.
Info: BKS Software, Guerickestr. 27, 1000 Berlin 10, Tel.: (030) 3423066/67

Interaktives Programmiersystem Amber/2 für OS/2

Die Amber Software Deutschland, 100%ige Tochter von Amber Software International (USA) und Vertreiber von Programmiersystemen der 4. Generation in deutscher Sprache, kündigte soeben das neue Entwicklungssystem Amber/2 an. Diese auf Wörterbuchebene geschriebene Programmiersprache (mit relationalem Datenbanksystem) ist vollkommen kompatibel zu der Version Amber/-Plus, die für MS-DOS-Systeme erfolgreich in vielfachen Praxislösungen eingesetzt ist. Damit werden Softwareentwickler und Anwender in die Lage versetzt, die überaus vielseitigen und umfangreichen Anwendungslösungen im kommerziellen und technisch-wissenschaftlichen Bereich, die unter dem Betriebssystem MS-DOS verfügbar sind, auch unter OS/2 einzusetzen.

Praxiserprobte Anwendungslösungen für OS/2-Systeme

Die neue Version Amber/2 stellt wie das praxiserprobte Entwicklungs-System Amber/Plus eine natürliche Sprache der 4. Generation dar, die in ein unabhängiges, interaktives Programmier-System für die Entwicklung kommerzieller Applikation integriert ist.

Softwarehäuser jeder Größe können mit Amber/2 nun die gesamte Palette ihrer Anwendungslösungen, die praxiserprobt auf MS-DOS-Systemen im Einsatz sind, auch für OS/2- und PC/UNIX-Systeme anbieten. Die Umsetzung auf Amber/2 erfolgt in einer einfachen und leicht durchführbaren Konvertierung.

Der besondere Vorteil von Amber/2, dies gilt auch für Amber/Plus, liegt für den Softwareentwickler in der leicht verständlichen interaktiven Software-Umgebung. Amber unterscheidet sich von traditionellen Programmierungstools durch die Konstruktion in Form eines elektronischen Wörterbuchs in deutscher Sprache. Zudem ist Amber/2 eine objektorientierte Programmiersprache. Damit wird es dem Programmierer oder Computer-Benutzer ermöglicht, seine Anweisungen und Begriffe in leicht erlernbarer Art und Weise in das System einzugeben. Für die Erstellung von kommerziellen Programmen mit Amber/Plus oder Amber/2 benötigt daher ein Softwareentwickler nur etwa 10 bis 20% des Zeitaufwands gegenüber den traditionellen Methoden.

Amber/2 ist durch die Programmierung in C universell und zukunftsorientiert ausgerichtet. Die komplette interne Programmierung des Entwicklungssystems Amber/2 in der Programmiersprache C bildet eine universelle und höchst aktuelle Brücke zu der PC/UNIX-Welt.
Info: Amber Software Deutschland, Hansa Allee 2, 4000 Düsseldorf 11, (0211) 589056

QuickBasic-Spezialist Zoschke unter neuer Adresse

Insbesondere QuickBasic-Anwendern ist das Ing.-Büro Zoschke (Schönberg/Holst.) seit Jahren ein Begriff. Ab dem 1.9.89 firmieren die Tool-Spezialisten unter Zoschke Data GmbH. Um dem wachsenden Umsatz gerecht zu werden, wurden gleichzeitig neue Geschäftsräume bezogen.

Unter anderem ist Zoschke Data deutscher Exklusiv-Distributor der US-Software-Hersteller Crescent Software Inc. und MicroHelp Inc. Das »Flaggschiff« der Crescent-Linie ist QuickPak Professional, eine Toolbox für Microsoft QuickBasic mit über 400 Routinen. Von MicroHelp stammt unter anderem das Tool Stay-Res Plus, mit dessen Hilfe QuickBasic-Programmierer ohne intime PC-Kenntnisse speicherresidente Programme erzeugen können, die durch clevere Harddisk- oder EMS-Nutzung nur ca. 10 Kbyte Hauptspeicher belegen. Insgesamt hat Zoschke Data bereits mehr als zwanzig Toolboxen für Basic-Programmierer im Lieferprogramm; viele Pakete liegen auch in deutscher Version vor.
Info: Zoschke Data GmbH, Bahnhofstraße 3, 2306 Schönberg/Holstein, Tel.: (04344) 6166



Hard- und Software von Dell

Ab sofort ist bei der Dell Computer GmbH das Betriebssystem MS OS/2 in der Version 1.1 für die gesamte PC-Linie auf 80286- und 80386-Basis zu haben. Im Lieferumfang ist auch die grafische Benutzeroberfläche »Presentation Manager« enthalten. Die Dell-Adaption der Microsoft-Entwicklung bietet die Funktionen Speicherverwaltung, Multitasking, eine Schnittstelle zu Anwendungsprogrammen und ist abwärtskompatibel zu den meisten DOS-Anwendungen. Wie schon bei MS-DOS hat Dell sich nicht auf die bloße Übernahme des von Microsoft entwickelten Betriebssystems beschränkt. Die Version »Dell Enhanced MS OS/2 1.1« zeichnet sich durch verbessertes Multitasking sowie eine »Dual boot«-Funktion aus, die es möglich macht, von derselben Festplatte aus wahlweise MS-DOS oder MS OS/2 zu starten.

»Das Betriebssystem OS/2 mit dem Presentation Manager bietet unseren Kunden mehr Komfort und Flexibilität bei den Anwendungen«, kommentierte Dell-Geschäftsführer Michael P. Ammel die Markteinführung. »Es ist für alle Anwender, die ihren Dell-PC noch produktiver nutzen wollen, die ideale Plattform.« MS OS/2 Version 1.1 kostet bei Dell DM 675,- (+ MWSt.) und ist ab sofort erhältlich.

Dell-Hardware

Seit Frühjahr 1988 gibt es das System 310 mit 80386-Prozessor von Dell. Die Standardkonfiguration umfaßt VGA-Grafik (wahlweise farbig oder monochrom), 1 Mbyte-Hauptspeicher (aufrüstbar auf 16 Mbyte auf dem Systemboard, davon werden 8 Mbyte direkt auf dem Board und weitere 8 Mbyte auf dem 32-Bit-Bus installiert), 40 Mbyte-Festplatte (optional bis 322 Mbyte), 1,44-Mbyte-Diskettenlaufwerk, den Monitor, ein hochwertiges Keyboard mit 102 Tasten, eine serielle und zwei parallele Schnittstellen, sechs freie Steckplätze und die Diagnosesoftware »Dell System Analyzer«. Varianten und Spezialversionen nach Kundenwunsch, wie beispielsweise Netzwerkkarten, Tape-Streamer oder ein 80387-Coprozessor beziehungsweise ein Weitek-Coprozessor sind selbstverständlich möglich.

Einen flexiblen und kostengünstigen Einstieg in die Welt der 32-Bit-Anwendungen eröffnet Dell mit dem Dell System 316. Dieses PC-System basiert auf dem von Intel entwickelten Mikroprozessor 386SX, der, ebenso wie der 80286-Prozessor, mit einem 16-Bit-Bus versehen ist, jedoch echte 32-Bit-Verarbeitung erlaubt. Damit ist der Einsatz von 32-Bit-Software für 80386-Systeme zu einem niedrigen Hardware-Preis möglich. Das neue Dell-System verkörpert überdies ein neues Niveau der Eigenentwicklung: Das System 316 ist mit einem hauseigenen ASIC (application-specific integrated circuit) versehen. Darin sind serielle und parallele Ausgänge sowie die Tastatur-Schnittstelle auf dem Systemboard vereinigt.

Auf dem Markt der 386SX-Systeme zeichnet sich das Dell System 316 durch sieben freie Steckplätze als besonders ausbaufähig aus, überdies ist mit 644 Mbyte eine riesige Festplattenkapazität erreichbar. In der Grundkonfiguration ist das Dell System 316 mit einer 40 MB-Festplatte mit 29 ms Zugriffszeit ausgestattet. Der Hauptspeicher (1 Mbyte DRAM, davon 384 Kbyte für Fast Video und FAST BIOS) ist auf der Hauptplatine auf 8 Mbyte ausbaufähig. Individuelle Konfigurationen sind, wie bei allen Dell-Rechnern, in weitem Rahmen möglich.

Das Flaggschiff der Produktpalette ist das System 325, ein mit echten 25 MHz getakteten Rechner auf der Basis eines 80386-Prozessors. Das System 325 zeichnet sich durch 32-Bit-Architektur, Cache-Controller 82385 und Speicherzugriff im Page-Mode aus. Die Standardkonfiguration umfaßt VGA-Grafik (wahlweise farbig oder monochrom), 4 Mbyte Hauptspeicher (aufrüstbar auf 16 Mbyte), einen 32 Kbyte großen

Cache-Speicher in Verbindung mit einem Intel-82385-Cache-Controller, eine ESDI-Festplatte mit wahlweise 90 Mbyte, 150 Mbyte, 322 Mbyte oder 610 Mbyte, ein Diskettenlaufwerk, wahlweise 5,25"/1,2 Mbyte oder 3,5"/1,44 Mbyte, einen 14"-Monitor, ein hochwertiges Keyboard mit 102 Tasten, eine parallele und zwei serielle Schnittstellen, sechs AT- und zwei XT-Steckplätze sowie ein 32-Bit-Steckplatz für das optionale Speichererweiterungsboard sowie die Diagnosesoftware »Dell System Analyzer«.

Das System 325 kann unter MS-DOS, Dell-OS/2 und XENIX 386 gefahren werden. Alle drei Betriebssysteme lassen sich gleichzeitig auf die Festplatte laden und wahlweise booten. Durch sein schnelles Antwortverhalten eignet sich das System 325 ideal als starker Server in modernen Netzumgebungen.

Neue Grafikkarte und Monitore für Dell-Computer

Für alle Dell-Anwender, die ihre PCs für anspruchsvolle Grafik-Aufgaben einsetzen, bietet der Hersteller jetzt eine GPX-Erweiterungskarte in mehreren Versionen für noch höhere Auflösung, optimierte Farbdarstellung und schnelleren Bildschirmaufbau. Das Leistungspaket ist ebenso für CAD/CAM/CAE-Anwender von Vorteil wie auch für Desktop-Publishing und aufwendige Spreadsheet-Anwendungen.

Die Dell GPX-Karte basiert auf dem mit 50 MHz getakteten grafischen Coprozessor TMS34010. Dieser speziell für Grafik-Anwendungen entwickelte Coprozessor entlastet die Zentraleinheit des Rechners, wodurch die Ablaufgeschwindigkeit erhöht und Wartezeiten verringert werden. Gegenüber herkömmlichen VGA-Systemen kann die Geschwindigkeit bis auf das Zehnfache gesteigert werden. Die Beschleunigerkarte paßt an den gängigen ISA-Bus und stellt sich automatisch auf 16- oder 8-Bit-Verarbeitung ein.

In der Version GPX-1024/16 steht ein Video-RAM (VRAM) von 512 Kbyte zur Verfügung. Damit lassen sich bei einer Auflösung von 1024 x 768 Bildpunkten gleichzeitig 16 Farben darstellen. Bei einer Auflösung von 800 x 600 oder 640 x 480 Bildpunkten sind 256 Farben verfügbar.

Noch mehr Leistung bietet die Version GPX-1024/256 mit einem 1 Mbyte VRAM. Sie erlaubt auch bei der Höchstauflösung von 1024 x 786 Bildpunkten die Darstellung von 256 Farben sowie von Schattierungen. Damit sind auch 3-D-Grafiken und eine geradezu »fotorealistische« Bildqualität möglich.

Alle GPX-Karten von Dell unterstützen wichtige Hochleistungsprogramme wie zum Beispiel Microsoft Windows. Optional können die GPX-Karten mit einem Zusatzmodul für VGA ausgestattet werden, so daß auf einem System mit einem Monitor sowohl aufwendige Grafik-Programme im GPX-Modus als auch herkömmliche MS-DOS-Programme im VGA-Modus, also beispielsweise Textverarbeitung oder Datenbanken, gefahren werden können.

Um die volle Leistung der GPX-Grafikkarte zu erschließen, bietet Dell ferner eine Reihe hochauflösender Grafikmonitore mit Bilddiagonalen von 16" und 19" an.

Info: Dell Computer GmbH, Siemensstraße 32, 6070 Langen, Telefon: 0 61 03/70 10

Neue Digital-PCs

Digital Equipment führte unlängst in Deutschland die Familie der neuen Digital-PCs ein. Die Personalcomputer DECstation 200, 300 und 350 basieren auf den Mikroprozessoren Intel 80286, 80386SX bzw. 80386 und sind kompatibel zum Industriestandard.

Auf den DECstation-PCs laufen sämtliche heute verfügbaren MS-DOS-Anwendungen. Zudem sind sie offen für bereits erhältliche und zukünftige Anwendungssoftware unter den Betriebssystemen MS-DOS, MS-OS/2, Unix, Xenix und anderen.

Die DECstation 350 als Spitzenmodell der DECstation-PC-Serie verfügt über einen mit 20 MHz getakteten 32-Bit-Mikroprozessor Intel 80386, VGA-Grafik sowie ein leistungsfähiges 32-Bit-Bussystem. Zwei Mbyte Hauptspeicher, ein 3 1/2-Zoll-Diskettenlaufwerk mit 1,44 Mbyte und eine 80-Mbyte-Festplatte sind Standard. Als Option sind ein zusätzliches 5 1/4-Zoll-Diskettenlaufwerk mit 1,2 Mbyte Speicherkapazität sowie ein Magnetbandsystem erhältlich. Vier freie Erweiterungs-Steckplätze ermöglichen einen flexiblen Ausbau. Die DECstation 350 kostet ab 16.100,- DM.

Info: Digital Equipment GmbH, Freischützstr. 91, 8000 München 81, Tel.: (089) 95910

AST 33-MHz-Rechner jetzt verfügbar

Früher als erwartet liefert AST Research seine ersten 33 MHz 80386-Rechner bereits ab August aus.

Der Rechner besitzt einen Intel 80386-Prozessor und arbeitet mit 33 MHz Null-Waitstate-Geschwindigkeit. Optimiert wird die Geschwindigkeit durch einen äußerst schnellen (25 ns) 32-Kbyte-Cache. Der Rechner ist zur Industriestandard-Architektur (ISA) voll kompatibel. Auf einer 32-Bit-Steckkarte befinden sich CPU, Cache, 2 Mbyte RAM (bis auf 4 Mbyte erweiterbar) und die Sockel für einen Intel 80387- sowie den Weitek 3167-Coprozessor.

Der RAM-Speicher kann mit zwei zusätzlichen 16-Mbyte-Speicherkarten auf insgesamt 36 Mbyte mit 80 ns SIMMs ausgebaut werden. Die Speicheraufrüstung kann in jeweils 1-Mbyte-Schritten vorgenommen werden, so daß der individuelle Speicherbedarf erreicht werden kann.

Der AST Premium 386/33 ist besonders für das Bearbeiten komplexer Aufgabenstellungen

wie CAD/CAM/CAE, Rechenanalysen, 3-D-Animationen oder umfangreiche Datenbanken oder als Server in lokalen Netzwerken und Multiuser-Umgebungen geeignet.

Drei Modelle umfaßt das 33-MHz-Programm:

Modell 5 ohne Festplatte für 15.868,- DM, Modell 105 mit 110-Mbyte-Festplatte (16 ms) für 20.651,10 DM und Modell 325 mit 320-Mbyte-Festplatte (16 ms) für 26.738,70 DM. Die Preise verstehen sich inklusive MwSt. Der Lieferumfang beinhaltet jeweils ein 5,25-Zoll-Diskettenlaufwerk, DOS 3.3, Tastatur sowie eine umfangreiche Benutzersoftware und Dokumentation.

Weiterhin besteht die Möglichkeit der Umrüstung zum 80486 Rechner mit 25 MHz durch das AST FASTboard 486/25 Board.

Info: AST Research Deutschland GmbH, Emanuel-Leutze Str. 1b, 4000 Düsseldorf 11, Tel.: (0211) 59570

Erster Wang-PC mit Microchannel-Architektur

Zusammen mit drei anderen, besonders preisgünstigen Industriestandard-Systemen auf Basis der 80286/80386SX Mikroprozessor-Technologie stellt Wang jetzt seinen ersten zur Microchannel-Architektur voll kompatiblen PC vor. Wie die Wang Laboratories Inc., Lowell/Mass., mitteilt, unterstützen die neuen PCs das Betriebssystem MS-DOS 4.01 und optional auch MS-OS/2 1.1 (einschließlich Presentation Manager).

Zu den vier neuen Wang-PCs gehören:

• der PC 250/16, ein Kompakt-PC auf Basis des Intel 80286 bei einer Leistung von 16 MHz zum Preis von 12 MHz;

• der PC280/20, ein Hochleistungs-PC mit 20 MHz, der als Einstiegs-Server für ein Local Area Network (LAN) oder als aufrüstbarer High-End-Arbeitsplatz verwendet werden kann;

• der PC350/165, ein kompaktes 80386SX-System, auf dem 32-Bit-Software wesentlich kostengünstiger läuft als auf 80386-System;

• der MC350/16S, Wangs Einstieg in die Microchannel-Architektur auf Basis des 80386 SX-Prozessors, dessen wichtigste Kennzeichen größere und schnellere Festplattenlaufwerke, mehr eingebaute Speicherkapazität und mehr Aufrüstmöglichkeiten als vergleichbare Produkte sind.

Die Basiseinheit der jeweiligen CPU einschließlich 1 Mbyte Speicher, Tastatur, eingebautem 4fach Disketten-/Festplatten-Controller sowie 1,2 Mbyte oder 1,44 Mbyte Diskettenlaufwerk kostet ab 4.200,- DM für den PC250/16, 5.500,- DM für den PC 280/20 und 5.300,- DM für den PC 350/16S. Alle Systeme sind sofort lieferbar.

Die preisgünstigste Konfiguration eines MC350/16S mit 2 Mbyte Speicher, Tastatur, eingebautem Disketten-/Festplatten-Controller, 1,44 Mbyte/3,5-Zoll-Diskettenlaufwerk und eingebautem 16-Bit VGA-Controller ist für 6.900,- DM erhältlich. Der MC 350/16S wird ab Oktober geliefert.

Info: Wang Deutschland GmbH, Lyoner Str. 26, 6000 Frankfurt/Main 71, Tel.: (069) 66750

Erweiterung der PC-Palette von Philips

Der neue 386er P 3360 von Philips ist mit 25 MHz getaktet. Der Hauptspeicher wird standardmäßig mit 4 Mbyte geliefert und ist auf dem Mainboard bis 8 Mbyte, mit 32-Bit-Speicherkarten bis zu 24 Mbyte erweiterbar. 64 Kbyte Cache-Memory erlauben einen sehr schnellen Speicherzugriff des Prozessors. Zwei serielle und eine parallele Schnittstelle sowie VGA-Karte und erweiterte Tastatur sind integriert. Es stehen zwei 32- und fünf 16-Bit-AT-Steckplätze sowie ein 8-Bit-XT-Steckplatz für Systemerweiterungen zur Verfügung. Der P 3360 kann standardmäßig mit einer 160- oder einer 338-Mbyte-Festplatte (18 ms) oder als Diskless-PC geliefert werden.

Mit einem 33 MHz getakteten 80386-Prozessor ist der P 3370 das Spitzenmodell unter den Philips-PCs. Das Tower-Chassis des P 3370 bietet ein Maximum an Flexibilität für Systemerweiterungen. Die Standard-Speicherkapazität des PC beträgt 4 Mbyte und ist auf der Hauptplatine auf 8 Mbyte (gesamt 24 Mbyte) erweiterbar. Die Festplatten haben Kapazitäten von 160 Mbyte bis zu 338 Mbyte und erlauben einen schnellen Zugriff auf den Online-Massenspeicher. Insgesamt verfügt der 33-MHz-Rechner über einen XT- und neun AT-kompatible Erweiterungsteckplätze. Ferner stehen zwei serielle und eine parallele Standardschnittstelle zur Verfügung.

Alle PCs arbeiten mit dem Betriebssystem MS-DOS 4.01 von Microsoft. Wahlweise lassen sich auch MS OS/2 Presentation Manager, LAN-Manager, Windows/386 einsetzen. In den zwei bzw. drei Fronteinschüben können sowohl Streamer- als auch CD-ROM-Laufwerke eingebaut werden.

Info: Philips Kommunikations Industrie AG, Weidenauer Str. 211-213, 5900 Siegen, Tel.: (0271) 4040

Sharp Color-Portable marktreif

Anfang 1990 beginnt der Vertrieb des ersten 386-Portable-Computers PC-8041 von Sharp mit 14-Zoll-Farb-LCD-Bildschirm. 512 Farben vermag das hintergrundbeleuchtete Farb-LC-Display bei der Auflösung von 640 x 480 Bildpunkten darzustellen. Der in DST-(Double Layered Supertwisted Nematic) LCD-Technik konzipierte 14-Zoll Color-Flachbildschirm ist voll VGA-kompatibel. Das Herz des Rechners ist ein mit 20 MHz getakteter 32-Bit 80386-Mikroprozessor, dem bei

Bedarf noch ein Coprozessor 80387 zur Seite gestellt werden kann.

Erstaunlich ist die extrem kurze Zugriffszeit der 40 MByte fassenden Festplatte mit nur 19 ms. Ein 3,5-Zoll-Diskettenlaufwerk (1,44 Mbyte) steht ebenfalls zur Verfügung. Der PC-8041 wird standardmäßig mit 2 Mbyte Hauptspeicher angeboten, der bis zu 8 Mbyte ausbaubar ist. Für Erweiterungen bietet er zwei Expansionslots.

Info: Sharp Electronics GmbH, Sonninstr. 3, 2000 Hamburg 1, Tel.: (040) 237750



Neue Laptops von Toshiba

Die Toshiba-Laptops T16000 und T3100e werden ab sofort auch in einer Version mit 40-Mbyte-Festplatte erhältlich sein. Die modifizierten Modelle tragen die Bezeichnung T16000/40 und T3100e/40.

Die 40-Mbyte-Festplatte in beiden Computern arbeitet außergewöhnlich schnell: Sie gewährleistet eine durchschnittliche Zugriffszeit von weniger als 29 bzw. 25 ms. Darüber hinaus bietet sie eine Speicherkapazität, die eine problemlose Benutzung modernster Software sowie anspruchsvoller Betriebssysteme (OS/2) ermöglicht.

Der T16000/40 ist ein batteriebetriebener Laptop mit einem 12 MHz getakteten Prozessor. Besondere Merkmale sind ein abnehmbarer hintergrundbeleuchteter Supertwist-Bildschirm mit EGA-Grafik und ein 1 bis 5 Mbyte Arbeitsspeicher, der zu einer schnellen RAM-Disk konfiguriert werden kann. Die neue Toshiba MasTimePower-Funktion optimiert die Stromversorgung und garantiert eine längere Betriebsdauer der Batterien.

Zwei serielle und eine interne Toshiba-Schnittstelle sorgen für entsprechende Erweiterungsmöglichkeiten. Der T16000/40 zeichnet sich durch eine Resume-Funktion aus, die dem Anwender die Fortsetzung seiner Arbeit an gleicher Stelle im Programm nach Wiedereinschalten des Gerätes ermöglicht. Das Gewicht des Gerätes beträgt 5,2 Kilogramm.

Der T3100e/40 ist ein netzabhängiger, auf einem 80286-Prozessor basierender Laptop. Das 5,9 Kilogramm leichte Gerät verfügt aber über einen internen Erweiterungsteckplatz (halbe Länge 8-Bit), gemäß Industriestandard, der jedoch auch für Toshiba-Karten zu nutzen ist.

Der Laptop hat einen Plasmabildschirm mit einer Auflösung von 640x400 dpi in dem Toshiba-Grafik-Modus sowie einen auf der Hauptplatine integrierten CGA-Grafikadapter. In der Grundausstattung verfügt das Gerät über einen 1 Mbyte Arbeitsspeicher, der bis zu 5 Mbyte erweitert werden kann.

Diese Erweiterung der Produktpalette entspricht der erfolgreichen Toshiba-Strategie, für jeden individuellen Anwenderbedarf entsprechende Geräte anzubieten. Das Toshiba-Laptop-Angebot umfaßt vom T1000 bis T5200/100 inzwischen zwölf Modelle.

Über den Toshiba Computer-Fachhandel ist der T1600/40 zu einem Preis von 11.377,- DM und der T3100e/40 zu einem Preis von 10.237,- DM erhältlich (unverbindlich empfohlene Preise inklusive Mehrwertsteuer).

T3200SX

Mit der Ankündigung des T3200SX stellt Toshiba erneut seine Kompetenz auf dem Gebiet der Laptop-Computer unter Beweis.

Der T3200SX, der die Produktlinie der netzabhängigen Toshiba-Laptops ergänzt, besitzt einen mit 16 MHz getakteten 80386SX Prozessor, VGA-Grafik, einen 1 bis 13 Mbyte Arbeitsspeicher, eine schnelle 40-Mbyte-Festplatte und interne Erweiterungssteckplätze nach dem Industriestandard.

Der T3200SX stellt eine preislich interessante Alternative für den Anwender dar, der die nächste Generation der 32-Bit-Software nutzen möchte. Seine großzügige Ausstattung entspricht den hohen Anforderungen der zukunftsweisenden Multitasking-Betriebssysteme OS/2 und UNIX.

Bei einer Größe von 370x395x99 Millimetern und einem Gewicht von 7,9 Kilogramm bietet er ein erstaunliches Maß an Leistung und Komfort. Er zeichnet sich durch

einen großen VGA-Bildschirm mit 16 Graustufen, eine Tastatur mit separatem Zehnerblock, mehrere interne Erweiterungssteckplätze und einen kompletten Satz an Schnittstellen, darunter auch zwei serielle Schnittstellen, aus.

Die internen Erweiterungssteckplätze ermöglichen dem Anwender die individuelle Anpassung für den eigenen Bedarf: Zwei Steckplätze stehen für Steckkarten nach dem Industriestandard zur Verfügung. Einer für eine lange 16-Bit- oder 8-Bit-Karte, der andere für eine kurze 8-Bit-Karte. Der kurze Steckplatz kann auch für Toshiba-Karten genutzt werden wie beispielsweise SYSLINK E, ein lokales Netzwerk, das auf einer Ethernet-Spezifikation basiert. Ein weiterer Steckplatz ist dem Toshiba-Modem vorbehalten. Sollten mehr als die internen Steckplätze benötigt werden, ist die Installation einer externen Erweiterungsbox für zwei oder fünf zusätzliche Karten möglich.

Der T3200SX reiht sich nahtlos in die Produktpalette der 286er sowie 386er Laptops zwischen dem T3200 und dem T5100 ein. Toshiba besitzt somit ein komplettes Angebot an Laptop-Computern, die aufgrund ihrer Leistung, Kompaktheit und Erweiterungsmöglichkeiten eine attraktive Alternative zu den traditionellen Desktop-Computern darstellen.

Der T3200SX wird über den Toshiba Computer-Fachhandel zu einem Preis von 14.797,- DM (unverbindlich empfohlener Preis inklusive Mehrwertsteuer) vertrieben.

Info: Toshiba Informationssysteme GmbH, Görlitzer Str. 5-7, 4040 Neuss 1, Tel.: (02101) 1370

Der Macintosh IIci

Der Macintosh IIci – das ist hohe Leistungsfähigkeit und verbesserte Funktionalität im Design des IIcx. Benutzer, die eine schnelle Programmausführung für speicherintensive Rechenblätter, Datenbanken und grafisch aufwendige Anwendungsprogramme benötigen, werden die deutlich verbesserte Leistungsfähigkeit des Macintosh IIci besonders schätzen.

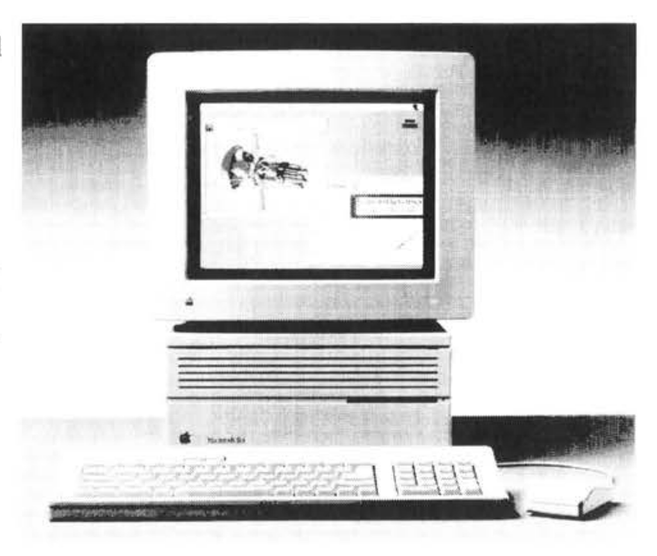
Der mit 25 MHz getaktete Motorola 68030-Mikroprozessor des Macintosh IIci trägt am deutlichsten zu dieser Leistungssteigerung bei. Durch die Erhöhung der Taktfrequenz arbeitet das System um etwa 45 Prozent schneller als der Macintosh IIcx oder IIx. Zur Beschleunigung komplexer mathematischer Berechnungen verfügt das System standardmäßig über einen 68882 Coprozessor. Durch die Installation einer Cache-Speicherkarte wird das System um weitere 20 bis 30 Prozent beschleunigt. Daraus ergibt sich eine maximale Gesamtleistungssteigerung von etwa 75 Prozent.

Der Macintosh IIci beinhaltet verschiedene Gestaltungsmerkmale, die die Möglichkeiten der 25 MHz 68030/68882-Kombination optimieren. Damit ist das System in der Lage, Speicherinhalt schneller und effizienter zu lesen als in vorhergehenden Architekturen.

Mehr Flexibilität

Der Macintosh IIci vereinigt eine Reihe von neuen Möglichkeiten in seiner Standard-Konfiguration, die dem Benutzer mehr Flexibilität an die Hand geben. Der eingebaute Videoanschluß ermöglicht es dem Macintosh IIci mit drei verschiedenen Bildschirmtypen zu arbeiten. Dabei handelt es sich um einen hochauflösenden Monochrombildschirm (12 Zoll) mit maximal 256 Graustufen, dem hochauflösenden AppleColor RGB-Bildschirm (13 Zoll) mit maximal 256 Farben oder Graustufen sowie dem Monitor ApplePortrait (15 Zoll) mit maximal 16 Graustufen.

Der Macintosh IIci verfügt über einen selbstkonfigurierenden eingebauten Video-Anschluß, dessen Vorteil sich beim Aufstellen der Anlage als augenscheinlich erweist. Der Benutzer muß lediglich noch einen Video-



Stecker in den dafür vorgesehenen Anschluß einstecken. Dies stellt verbesserte Erweiterungsmöglichkeiten dar, da im NuBus-Steckplatz keine Videokarte mehr zu installieren ist. Reduzierte Anschaffungskosten für das System sind eine weitere Konsequenz.

Der Macintosh IIci verfügt standardmäßig über 512 Kbyte ROM. Zusätzlich befindet sich ein ROM-SIMM-Sockel auf der Hauptplatine, der die Installation von ROM-Erweiterungen zu einem späteren Zeitpunkt erleichtert. Mit der 32-Bit-Adressierung kann das Betriebssystem 4 Gbyte Speicher ansprechen. Durch das hierarchische Dateisystem kann eine Dokumentenverwaltung einfacher organisiert und der Zugriff auf Dateien erleichert werden. Das ROM beinhaltet die Unterstützung für 32-Bit-Farbgrafik mit QuickDraw, die es den Farb-Systemen erlaubt, gleichzeitig bis zu 16 Millionen Farben gleichzeitig zu zeigen.

Hardware-Eigenschaften

Die Hardware-Eigenschaften stimmen grundsätzlich mit denen des IIcx überein. Hierzu gehören drei NuBus-Erweiterungssteckplätze, ein eingebautes 3,5-Zoll-Festplattenlaufwerk, sieben Standardanschlüsse für Peripheriegeräte, Erweiterungsmöglichkeiten für das RAM auf maximal 8 Mbyte. Der Macintosh IIci ist mit einem Apple SuperDrive-Laufwerk für HD-Disketten mit 1,44 Mbyte ausgestat-

tet. Mit diesem Laufwerk können 3,5-Zoll-Disketten verwendet werden, die für den Macintosh oder für verschiedene andere Personalcomputer initialisiert wurden. Das System verfügt über die gleichen Anschlüsse wie der IIcx und dazu noch den Video-Anschluß zur Unterstützung des eingebauten Video.

Die Macintosh-Betriebssystemsoftware besteht aus der Systemdiskette Version 6.0.4., der Druckerdiskette (Druckertreiber für alle Apple-Drucker) sowie Dienstprogrammdisketten (mit den Dienstprogrammen wie z.B. »Dateien konvertieren« oder »Festplatte installieren«). Mitgeliefert wird die HyperCard Version 1.2.5. Mit dem Macintosh IIci kompatibel und optional erhältlich ist das Betriebssystem Unix (die A/UX-Version 1.1.1).

Der Macintosh IIci ist in der englischen Version sofort verfügbar. Die deutsche Version ist ab November über die Apple Vertragshändler zu beziehen. Er wird in der Grundausstattung mit einer eingebauten 3,5-Zoll-Festplatte mit 80 Mbyte Speicherkapazität geliefert. Der Preis für diese Konfiguration wird bei etwa 18.000,- DM (zuzüglich MwSt) liegen.
Info: Apple Computer GmbH, Ingolstädter Str. 20, 8000 München 45, Tel.: (089) 350340

Hochauflösender Bildschirm mit OS/2 PM-Unterstützung

Die Ventek Corporation kündigt die Unterstützung vom OS/2 Presentation-Manager für Ihre AT- und PS-Bus 20 Zoll hochauflösende Monochrombildschirme an. Ventek Corporation wird zusammen mit ihrem deutschen Exklusiv-Repräsentanten, Intelekta, auf der Systems '89 die Leistungsfähigkeit dieser neuen Implementierung demonstrieren.

Venteks Subsysteme sind die einzigen Ultra-High-Resolution 20-Zoll-Bildschirme die dem Anwender vollständige Software-Kompatibilität und die Fähigkeit der Darstellung von 64 Graustufen bieten, die von den meisten VGA-Applikationen verlangt werden. Diese wird dadurch möglich, daß Ventek eine komplette Reihe von Software Treibern für Ultra-High-Resolution-Support von Windows, GEM, Ventura Publisher, WORD 5.0, AutoCAD, Lotus 1-2-3 und vielen anderen Publishing-, Image Processing- und CAD-Programmen mitliefert. Außerdem sind Ventek Monitore völlig kompatibel zum IBM VGA-Standard auf Registerlevel, hierzu benutzt Ventek analoge Graustufen-Monitore.

Laserdrucker und Scanner von Canon

Die Laserdrucker-Serie von Canon wird durch ein neues Modell ergänzt: den LBP-4. Bei diesem Modell handelt es sich um einen Laserstrahldrucker im unteren Preisbereich. Die Druckleistung beträgt 4 Seiten/Minute im Kopiermodus. Da Maße und Gewicht des neuen Laserstrahldruckers LBP-4 etwa 50 Prozent des LBP-8III entsprechen, hat dieser auch eine neue Toner-Cartridge mit einer Kapazität von 3.500 Seiten. Der Toner wurde dahingehend modifiziert, daß kein umweltschädigendes Ozon erzeugt wird. Die zusätzlich erhältliche Papierzuführung erhöht die serienmäßig vorhandene Papierkapazität von 50 auf 200 Blatt. Der Drucker verfügt über einen Speicher von 512 Kbyte und die komplette Intelligenz des Canon LBP-8III. Skalierbare Fonts, Rotation, Spiegelung, VDM-Mode, um nur einige Merkmale zu nennen, weist der Canon LBP-4 bereits in der Standardversion auf. Eine Speichererweiterung mit 1 Mbyte ist auf Platine, eine weitere mit ebenfalls 1 Mbyte als »Huckepack«-Erweiterung verfügbar. Fontkarten können aus der 8III-Serie verwendet werden.

Auch im Scannerbereich kann Canon anläßlich der Systems '89 eine Neuheit präsentieren: den Flachbettscanner IX-30F. Die wichtigsten Daten:

Auflösung von 75 bis 300 dpi einstellbar, Format DIN A4, 256 Graustufen, SCSI-Interface, Software »ScanDo«. Das SCSI-Interface wird in drei Konfigurationen, komplett mit Anschlußkabel, angeboten: PC-, Microchannel- und Apple-Version. Ein Einzelblatteinzug ist als Zubehör erhältlich.
Info: Canon Rechner Deutschland GmbH, Fraunhoferstraße 14, 8033 München-Martinsried, Tel.: 089/85 70 01-0

Kurzweil Lesesysteme K 5100 und Discover Freedom

Die Modelle der Kurzweil-Systemfamilie lesen ohne Lernphase alle lateinischen Schriften (auch proportionale, wie z.B. Satzschriften) automatisch ein und verfügen über Konvertierungsroutinen zu gängigen Textverarbeitungs- und Grafikprogrammen, zu Desktop Publishing und Datenbanken. Die Schriftenerkennung läuft unabhängig vom PC, über die eingesetzte Coprozessorplatine (32-Bit-Prozessor).

Zur Systems '89 stellt CCS Compact Computer Systeme GmbH, Hamburg – General-Repräsentant der Kurzweil-Lesesysteme – zwei neue Produkte vor: Das neue Spitzenmodell der Systemfamilie K 5100 basiert hardwareseitig auf dem bekannten Discover Modell 40 und arbeitet ebenfalls mit einem 400 dpi Spezialscanner, mit dem Schriftgrößen von 6 bis 24 Pica-Punkt sicher gelesen und Halbtonbilder in 64 Graustufen erfaßt werden können.

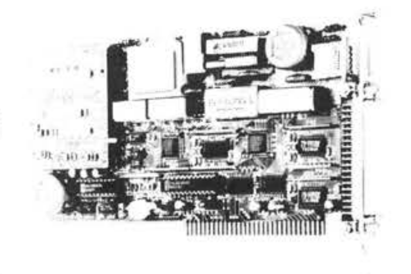
Besonderes Merkmal des neuen ICR Lesesystems K 5100 ist die Möglichkeit der interaktiven Zeichenkorrektur. Mit diesem neuen Softwaremodul werden zunächst fehlerhaft er-

kannte Zeichen manuell verbessert und nachtrainiert. Kurzweil bietet damit in Ergänzung der automatischen Schrifterkennung nun auch die Verarbeitung schwer lesbarer Zeichen sowie die Erkennung verschiedener Sonderzeichen. Erweiterte Zeichenerkennungsroutinen ermöglichen auch die Verarbeitung schwieriger Vorlagen, wie z.B. Dokumente auf dünnem Papier.

Das zweite neue Produkt ist das Kurzweil-Modell Discover Freedom, ein scannerunabhängiges ICR-Lesesystem mit Scannertreibern für Datacopy (730GS, 730, 830, Jetreader) Hewlett Packard (Scanjet, Scanjet Plus), Microtek (MS300 A), Dest (PC Scan). Treiber für Siemens- und Agfa-Scanner sind in Vorbereitung. Kurzweil durchbricht mit diesem neuen Modell zum ersten Mal seine bisherige Produktpolitik einer strengen Kopplung der Lesesysteme an spezielle Kurzweil-Scanner. Kurzweil ICR Lesesysteme werden so für praktisch alle relevanten Scannerfabrikate für neue Anwenderkreise von größtem Interesse sein.

Kurzweil-Anwender können in Zukunft Dokumente als reine Grafikdateien einscannen und im TIFF-Format abspeichern. Unabhängig von der Dokumentenerfassung können jetzt die eingelesenen Seiten z.B. im Batchbetrieb auch zu einem späteren Zeitpunkt durch die ICR Software interpretiert werden. Bei der Umsetzung eingescannter Dokumente in Text-Dateien kann durch entsprechende Voreinstellung gleichzeitig auch eine Faksimile-Datei im TIFF-Format abgespeichert werden.

Standardmäßig enthalten Kurzweil Systeme in Zukunft Konvertierungssoftware für die direkte Textkonvertierung in DCA (UDF)-ASCII, WK1, ASCII, CEOWrite, RFT, DEC WPSPlus, Microsoft Word, Multimate, Samna Word, Sprint, Volkswriter, Wang IWP, WordMark, WordPerfect, Wordstar; als Option: Interleaf, FrameMaker.
Info: CCS, Hauptstr. 25, 7125 Bad Boll, Tel.: (07164) 2059



Faxkarte ohne Scanner und Drucker zugelassen

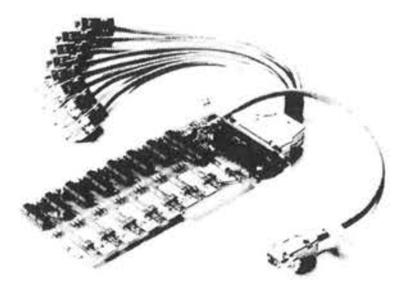
Die Deutsche Bundespost hat eine PC-Faxkarte für den Betrieb zugelassen, mit der frei von postalischen Beschränkungen ein Faksimile direkt aus dem PC an die weltweit installierten Faxgeräte versendet werden kann.

Damit gelang es der Dr. Neuhaus Mikroelektronik GmbH als erstem Unternehmen, eine Zulassung als CCT-Fax in dem sich liberalisierenden Endgerätemarkt zu erhalten. Die Zulassungsnummer lautet A200507X.

Mit folgenden Leistungsmerkmalen kann FAXY PC BASIS von Dr. Neuhaus aufwarten:

- Versenden von bis zu 16 Dateien.
- Normal- und Feinauflösung.
- Manuelle und automatische Betriebsart.
- Senden wahlweise mit 9600 oder 4800 Baud.
- Darstellung von Faxen auf dem Monitor mit Drehen um 180 Grad und Zoomen in drei Stufen.
- Empfangs- und Sendejournal.
- Vollautomatisches Empfangen mit 9600 Baud.
- Drucken von Faxdateien auf Nadel- und Laserdruckern.
- Jedes Fax wird auf der Festplatte gespeichert.

Durch den hohen Leistungsumfang und den günstigen Preis ab DM 799,- zzgl. MWSt. (je nach Leistungsumfang der mitgelieferten Software) kann zukünftig jeder über sein eigenes Personal-(PC-)Fax verfügen.
Info: Dr. Neuhaus GmbH, Haldenstieg 3, 2000 Hamburg 61, Tel.: (040) 553040



VGA Multi-Video-Karte

Für den Einsatz auf Messen, Präsentationen, Schulungen und zur Informationsvervielfachung wurde von MEZ, Beckum, eine Steckkarte für PCs entwickelt. Diese erlaubt überall dort, wo Bildschirminformationen von einem PC auf bis zu acht Monitoren übertragen werden sollen, eine Präsentation bis zum hochauflösenden 1024x768-VGA-Modus. Die VGA-MVK ist inklusive Verteileradapter für acht Monitore für knapp 1000,- DM beim Fachhandel erhältlich.
Info: MEZ Menke EDV Zubehör GmbH, Zementstr. 116, 4720 Beckum, Tel.: (02521) 700103

Exabyte-Laufwerk

Neu in Produktspektrum von Impec ist das Backup-System Exabyte. Hierbei handelt es sich um ein SCSI-Bandlaufwerk zur Datensicherung bis zu 2,3 Gbyte. Als Medium wird dazu ein handelsübliches Video-8-Band verwendet. Auch das Laufwerk stammt aus der Video-Branche und hat Helical Scan als Aufzeichnungsverfahren.

Für die Installation wird ein Controller benötigt, der für DOS- und PS/2-Rechner geliefert wird. Das Laufwerk macht schnelle Backups und Restores (90 Mbyte in ca. 45 Min.), und es können mehrere Datensicherungen auf einem Band hintereinander realisiert werden.
Info: Impec Computervertriebs GmbH, Waldhörnlestr. 18, 7400 Tübingen, Tel.: (07071) 70020

3Com präsentiert »Client-Server-System«

Der kalifornische Netzwerk-Hersteller 3Com kündigte unlängst weltweit das erste »Client-Server-System« – 3+Open CSS – in Abstimmung mit namhaften Software-Entwicklern, darunter Ashton-Tate, Lotus, Oracle und Microsoft an.

Unter diesem System versteht man das organische Zusammenspiel zwischen Netzwerk-Arbeitsplätzen auf Workstation-Basis (Clients) und Netzwerk-Zentralrechnern (Server). Dabei steht die wirtschaftlich sinnvolle Verteilung der Aufgaben zwischen den wichtigsten Netzwerk-Komponenten im Mittelpunkt. Das Auftrennen der Programmarbeit zwischen einer sogenannten »Front-End«-Komponente zur örtlichen Datenverarbeitung und einer »Back-End«-Komponente für umfangreiche Rechenoperationen, wie Datenspeicherung und Netzwerksicherheit, ist die Kern-Philosophie dieser neuen Technologie. Das »Client-Server-System« will Rechnerleistung, Anwenderprogramme und Datenpräsentation im Netzwerk aufgabenbezogen zwischen Netzwerk-Arbeitsplätzen, Servern, Minicomputern und Großrechnern aufteilen. Kennzeichnend dafür ist, daß rechenintensive Operationen, wie Datenbankaufgaben, Datei- und Druckerverwaltung und ähnliche Großrechner-Funktionen im Server ablaufen, während die Arbeitsplätze für aktuelle Anwenderarbeiten und Anfragen frei bleiben. Kurze Antwortzeiten und hohe Leistungsfähigkeit sollen die Nachteile des traditionellen Systems der gemeinsamen Dateinutzung überwinden.

Diese hohen Ansprüche sind nur durch eine absolute Offenheit des Netzwerk-Systems zu erreichen. So werden mehrere System-Protokolle aber auch unterschiedliche Betriebssysteme, wie DOS, OS/2, UNIX, Macintosh und andere unterstützt.

3+Open CSS ist ein voll ausgerüstetes Netzwerk-System, bestehend aus der 3+Open LAN-Manager-Betriebssystem-Software Version 1.1 mit DPA (Demand Protocol Architecture), dem neuen 3Com-Server auf 80386-Basis, den 3Station-Netzwerk-Arbeitsplätzen ohne Laufwerke und Programme für elektronische Post (E-Mail) und für Inter-Netzwerk-Verbindungen. Im System können viele bedeutende kommerzielle Programme für IBM-PCs, OS/2-Rechner, Macintosh-Computer und kompatible Systeme eingesetzt werden.

3Com weist besonders darauf hin, daß Digital Equipment-, Hewlett Packard- und IBM- sowie OSI-(Open System Interconnect)-Systeme am 3+Open CSS angeschlossen werden können.

Dazu dienen z.B. die X.25-Netzwerk-Bridge von 3Com, Gateways mit IBM-kompatiblem SNA-Aufbau und TCP/IP-(Transmission Control Protokol)-Dienste für DOS-Workstations. Neben der Leistungsverbesserung erhält der Anwender zusätzlich die Preisvorteile einer Gesamtlösung.

3+Open LAN-Manager Version 1.1

Diese erweiterte Version des vor einem Jahr eingeführten 3+Open LAN-Manager ist Bestandteil des »Client Sever System«. Die Demand Protocol Architecture (DPA) schafft mehr Möglichkeiten für verteilte Anwendungen. Das neue NetBios-Protokoll erhöht die freien Speicherkapazitäten auf der Workstation und mit dem Resident Protocol Manager ist es möglich, auf der Workstation mehrere Protokolle zu laden, mit Tastendruck zu wechseln oder wieder zu verlassen.

3+Open als Entry-Version für zehn Anwender

Das neue 3+Open LAN-Manager Entry System II schließt eine Lücke zwischen der bisherigen »Einstiegs«-Version für fünf Anwender und der großen Version für praktisch unbegrenzt viele Arbeitsplätze. Ausgestattet mit allen Merkmalen der Version 1.1 von 3+Open, verfügt dieses neue Netzwerk-Betriebssystem über alle Möglichkeiten, die sowohl die »Client-Server«-Umgebung als auch die PC-Host-Verbindung erfordern. Zudem unterstützt es alle zusätzlichen Dienste wie 3+Open Mail, 3+Open Internet, 3+Open LAN Vision.

3Server/500

Der mit einem großen Arbeitsspeicher (max. 16 Mbyte RAM) und bis zu 6 Gbyte Plattenspeicher ausgestattete Server arbeitet auf der Basis des 80386-Prozessors. Die Drei-Port-Architektur unterstützt verschiedene Bus-Strukturen, liefert einen höheren Datentransfer sowohl im Ethernet als auch im Token-Ring und AppleTalk. Er ist besonders für Einsätze im kommerziellen Bereich konzipiert, bei denen große Datenmengen zu bewältigen sind. Neben den erwähnten Protokollen unterstützt er auch SNA, DECnet, TCP/IP und die internationalen Standards X.400 und X.25.

3Station-Familie

Neben der seit 1988 zur 3Com-Produktpalette gehörenden 3Station/2E wurden zwei weitere Versionen angekündigt. Die 3Station/2ED kombiniert die IBM-PC-kompatible Architektur der 3Station/2E mit der Polygon-Software Polystar. Durch einfaches Umschalten kann der Netzwerk-Arbeitsplatz einmal als PC fungieren und zum anderen das hochauflösende DEC-Grafi-Terminal VT-340 emulieren. Die Version 3Station/2X bietet höhere Leistung und unterstützt mit seinen 3 Mbyte Arbeitsspeicher fensterorientierte Arbeitsumgebungen, die z.B. auf LIM 4.0 basieren.

Info: 3Com GmbH, Gustav-Heinemann-Ring 123, 8000 München 83, Tel.: (089) 678210

Mit Microsoft-Seminaren sicher in die Zukunft

Die Spezialseminare des Microsoft Instituts vermitteln in kleinen Gruppen intensiv all das, was zum Einstieg in die Programmentwicklung nötig ist. Modernste Trainingsmethoden sowie PC-Demonstrationen und -Übungen sind selbstverständlich. Professionelle Entwickler bekommen durch professionelle Schulung die Möglichkeit, ihren hohen Wissenstand den neuen Gegebenheiten anzupassen.

Das Microsoft OS/2-Einführungsseminar
Zweitägiges Seminar für PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache besitzen.

Datum	Tag
Düsseldorf	
06./07.11.89	Mo./Di.
München	
04./05.12.89	Mo./Di.

Der Microsoft OS/2-Workshop
Dreitägiges Seminar für PC-Software-Entwickler, die Programmiererfahrungen in einer höheren Programmiersprache und C-Kenntnisse besitzen sowie das MS-OS/2-Einführungsseminar besucht haben.

Datum	Tag
Düsseldorf	
08./09./10.11.89	Mi./Do./Fr.
München	
06./07./08.12.89	Mi./Do./Fr.

Das Microsoft Windows-Einführungsseminar
Zweitägiges Seminar für PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Datum	Tag
München	
06./07.11.89	Mo./Di.
11./12.12.89	Mo./Di.

Der Microsoft Windows-Workshop
Dreitägiges Seminar für PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das Microsoft Windows Einführungsseminar besucht haben.

Datum	Tag
Düsseldorf	
25./26./27.10.89	Mi./Do./Fr.
München	
08./09./10.11.89	Mi./Do./Fr.
13./14./15.12.89	Mi./Do./Fr.

Microsoft Presentation Manager-Einführungsseminar
Zweitägiges Seminar für Windows- und C-Programmierer.

Datum	Tag
Düsseldorf	
16./17.10.89	Mo./Di.
18./19.12.89	Mo./Di.
München	
27./28.11.89	Do./Fr.

Microsoft Presentation Manager-Workshop
Dreitägiges für PC-Software-Entwickler, die Programmiererfahrungen in C und eventuell in Windows besitzen sowie das Microsoft Presentation Manager Einführungsseminar besucht haben.

Datum	Tag
Düsseldorf	
18./19./20.10.89	Mi./Do./Fr.
20./21./22.12.89	Mi./Do./Fr.
München	
29./30.11/01.12.89	Mi./Do./Fr.

Microsoft LAN-Manager-Einführungsseminar
Zweitägige Seminar für Netzanwender und Netzwerk-Softwareentwickler mit Programmiererfahrung in C.

Datum	Tag
Düsseldorf	
23./24.11.89	Do./Fr
04./05.12.89	Mo./Di.
München	
13./14.11.89	Mo./Di.

Microsoft LAN-Manager-Workshop
Dreitägiges Seminar für PC-Softwareentwickler mit Programmiererfahrung in C, die am LAN-Manager Einführungsseminar teilgenommen haben.

Datum	Tag
Düsseldorf	
06./07./08.12.89	Mi./Do./Fr.
München	
15./16./17.11.89	Mi./Do./Fr.

Microsoft SQL-Server-Einführungsseminar
Zweitägiges Seminar für PC-Softwareentwickler. Kenntnisse von MS OS/2 und dem MS LAN-Manager sollten vorhanden sein, SQL- und C-Kenntnisse sind von Vorteil.

Datum	Tag
Düsseldorf	
11./12.12.89	Mo./Di.
München	
23./24.10.89	Mo./Di.

Microsoft SQL-Server-Workshop
Dreitägiges Seminar für PC-Softwareentwickler mit Kenntnissen in C und SQL, die am Einführungsseminar und Workshop für OS/2 und den LAN-Manager teilgenommen haben.

Datum	Tag
Düsseldorf	
13./14./15.12.89	Mi./Do./Fr.
München	
25./26./27.10.89	Mi./Do./Fr.

Microsoft Excel-Makro-Programmierung
Dreitägiges Seminar für PC-Software-Entwickler, die mit Microsoft Excel schon einigermaßen vertraut sind.

Datum	Tag
Düsseldorf	
15./16./17.11.89	Mi./Do./Fr.
27./28./29.11.89	Mo./Di./Mi.
München	
16./18./18.10.89	Mo./Di./Mi.









LAN-Software-Katalog

Wer seine Personalcomputer in einem Netzwerk zusammenschließt, muß auch netzwerkfähige Programme benutzen, die den Datenaustausch zwischen den Rechnern gewährleisten. Im neuen LAN-Softwarekatalog vom Hüthig-Verlag finden Sie auf fast 500 Seiten mehr als 350 Programme aus verschiedenen Anwendungsbereichen, die alle Anforderungen an eine netzwerkfähige Software erfüllen. Das Buch liegt nun in der dritten, aktualisierten Fassung vor. Das stark erweiterte Werk wurde nach der CeBIT 1989 aufgefrischt. Sie finden darin Softwarebeschreibungen aus folgenden Bereichen: LAN-Betriebssysteme (so auch der OS/2 LAN-Manager), Utilities, programmierbare Datenbanken, Textverarbeitungen und Pakete zur Büroautomatisierung. Außerdem Software zur Auftragsbearbeitung, betriebliches Rechnungswesen, Lohn und Gehalt, Fertigung, Lagerhaltung, CAD/CAM/CAE/CAT und unterschiedliche Branchenlösungen. Die Beschreibungen umfassen unter anderem die Kriterien Eignung, Zielgruppe, Funktionen, Schnittstellen, Preis sowie Hersteller und Vetrieb. Das Buch dürfte für diese Software die einzig erhältliche Marktübersicht sein.

Paulo Heitlinger/York Simon/Karin Brotz: LAN-Software Katalog. Heidelberg, Hüthig, 1989; 470 Seiten; ISBN 3-7785-1814-3; DM 88,-.

Der OS/2 LAN-Manager

Mit dem Einsatz des OS/2 LAN-Managers wird es erstmals möglich, die vielfälltigen Multitasking-Konzepte des Betriebssystems OS/2 auch als Multiuser-Umgebung zu nutzen. Dabei wird das bereits vielfach erfolgreich erprobte Prinzip der zentralen Datenverwaltung bei dezentraler Rechenkapazität der lokalen Netzwerke weiter perfektioniert. Mit dem Buch »Der OS/2 LAN-Manager« aus dem IWT-Verlag wird das Prinzip eines derartigen Einsatzes veranschaulicht und das Einsatzspektrum des OS/2 LAN-Managers abgesteckt. Es wendet sich dabei an alle, die bereits mit lokalen Netzwerken arbeiten oder die Leistungsfähigkeit des LAN-Managers kennenlernen wollen. Darüber hinaus stellt es aber nicht nur die darin integrierten Konzepte vor. Das Buch bietet zum einen dem Anwender eine vielfältige Bedienungsgrundlage und gibt zum anderen Programmierern, die sich mit netzwerkfähigen Applikationen beschäftigen, die notwendige Unterstützung, die Programmierschnittstelle des LAN-Managers effektiv zu nutzen und derartige verteilte Systeme marktgerecht zu konzipieren.

Armin Ertl: Der OS/2 LAN-Manager. Vaterstetten, IWT, 1989; 350 Seiten; ISBN 3-88322-240-2.

Der PC im Netzwerk

Umfassende Informationssysteme, zunächst nur als Insellösungen in Form von lokalen Netzen, sind im Begriff die Welt zu erobern. Der nächste Schritt bedeutet die Verbindung dieser Kommunikationsinseln über Gateways mit Großrechnern sowie Daten- und Wissensbanken. Die zahlreichen LANs verbinden sich zu einem Internet, in dem jeder Teilnehmer auf alle ihm darin verfügbaren Ressourcen zugreifen kann. Das derzeit größte Problem bei der Realisierung dieses Vorhabens ist die Inkompatibilität der verschiedenen Systeme. Die dabei auftretenden Standardisierungsprobleme werden in dem Buch »Der PC im Netzwerk« aus dem Vogel-Verlag erläutert. Auf 200 Seiten bietet es fachliches Grundwissen und stellt nicht nur technische Sachverhalte verständlich dar. So geht der Autor beispielsweise auch auf die möglichen Wirkungen integraler Informationssysteme auf die Gesellschaft und die daraus resultierenden Folgen ein. Im Anhang befindet sich ein Glossar sowie Tabellen mit Einheiten und Kürzel-Erläuterungen.

Ottokar R. Schreiber: Der PC im Netzwerk. Würzburg, Vogel, 1989; 220 Seiten; ISBN 3-8023-0262-1; DM 40,-.







SQL-Programmieren in Datenbanken

Structured Query Language, kurz SQL, vom amerikanischen Standardisierungsinstitut (ANSI) zum Standard für Datenabfragesprachen erklärt, hat sich schnell zur Lingua Franca im Computersektor etabliert. Relationale Datenbanken im Mainframe- und Minicomputerbereich sind nun auch als PC-Versionen verfügbar, wodurch SQL auch in diesen Bereich vorgedrungen ist. Eine praxisorientierte Dokumentation über diese Sprache stellt das Buch »SQL Programmieren in Datenbanken« aus dem Verlag McGraw-Hill dar. Es wendet sich an alle, die SQL als Mittel zum Zweck einsetzen, um eine bestimmte Aufgabe auszuführen: Benutzer von interaktivem SQL, wie auch Programmierer, die Embedded-SQL für Zugriffe aus Programmen heraus einsetzen. Alle wichtigen Befehle sind darin ausführlich beschrieben, so daß sich auch SQL-Anfänger gut zurechtfinden. Neben den im ANSI-Standard festgelegten Definitionen geht der Autor auch auf die wichtigsten Erweiterungen ein, die die verschiedenen relationalen Datenbankprodukte in der Regel bieten. Auch in diesem Werk finden Sie im Anhang ein Glossar und zu dem eine SQL-Kurzreferenz.

Frank Lusardi: SQL-Programmieren in Datenbanken. Hamburg, McGraw-Hill, 1989; 190 S.; ISBN 3-89028-166-4; DM 58,-.

Datenbankabfrage in SQL

Dieses Buch aus dem Markt & Technik Verlag zeigt, wie Sie mit SQL Daten in eine relationale Datenbank eingeben, ändern und löschen. In kompensierter Form werden alle nötigen Informationen geboten. Sie finden genaue Arbeitsanleitungen, wie Sie die Daten übergeben, wie sie angezeigt und wie die entsprechenden Reports angelegt und formatiert werden. Es setzt keine speziellen EDV-Kenntnisse voraus, allgemeines EDV-Wissen ist jedoch von Vorteil. Der Befehlssatz orientiert sich am Urvater der relationalen Datenbanken, dem Großrechnersystem DB2 von IBM. Dieses Buch ist auch für Profis geeignet, da es auch Themen wie Embedded SQL und Zugriffsgeschwindigkeit nach physikalischem Navigieren mit dem Datenpool behandelt. Im Anhang werden unter anderem die häufigsten Fehlerursachen analysiert. Der Autor gibt hier seine langjährige Erfahrung im Gebrauch mit SQL als Programmiersprache weiter. Der Syntax- und Kommando-Überblick sowie ein großzügiges Stichwortverzeichnis erleichtern den Umgang mit diesem Buch.

Peter Beger: Datenbankabfrage mit SQL. Haar, Markt & Technik, 1989; 330 Seiten; ISBN 3-89090-648-6; DM 69,-.

Computerschnittstellen

Der Hanser-Verlag hat ein Buch über die drei wichtigsten Computerschnittstellen Centronics, V.24 und IEC-Bus herausgebracht. Hier finden Hard- und Software-Versierte in verständlicher Sprache alle notwendigen Informationen.

Nach der Darstellung der verbreitetsten Druckerschnittstelle werden Schnittstellenleitungen und das Handshaking erläutert. Danach wird die technische Realisierung beschrieben. Den Abschluß dieses Abschnitts bilden die Programmiermöglichkeiten und ein Vorschlag für eine Druckeransteuerung.

Bei der V.24-Schnittstelle werden Signale, Leitungen, Pegel, Logikdefinitionen und die verschiedenen Übertragungsarten behandelt. Die IEC-Bus-Schnittstelle wird mit der Struktur des Interface-Systems, der IEC-Buslogik, der Handshake-Verfahren und der Gerätestruktur erläutert. Es folgen Schnittstellenrealisierung und Programmierung des IEC-Busses.

Die Programmbeispiele sind allgemein gehalten, sie sind nicht auf einen speziellen Rechnertyp festgelegt. Rund 50 Seiten beinhalten Listings in den Sprachen Basic, Pascal, C und Z80-Assembler. Genaue Beschreibungen der entsprechenden ICs im praktischen Einsatz runden das Buch ab.

Lothar Preuß/Harald Musa: Computerschnittstellen. München, Hanser, 1989; 230 Seiten; ISBN 3-446-15341-1; DM 78,-.

Eine komfortable Benutzeroberfläche in C (Teil 6):

Noch mehr Objekte für Ihre Dialogboxen

Nachdem in der vorangegangenen Ausgabe des *Microsoft System Journals* der Dialog-Manager und die beiden Dialogobjekte »Edit« und »Action Button« vorgestellt wurden, stehen diesmal zwei weitere skalare Objekte auf dem Programm: »Push-Buttons« und »Radio-Buttons«.

Neben alphanumerischen Eingabefeldern und Terminatoren stellen die Push-Buttons und Radio-Buttons zwei weitere elementare Dialogobjekte dar. Eingesetzt werden sie überall da, wo sich der Anwender für oder gegen eine bestimmte Auswahlmöglichkeit entscheiden soll.

Der Unterschied zwischen den beiden Dialogobjekten liegt in der Anzahl der Möglichkeiten, die dem Anwender bei seiner Auswahl zur Verfügung stehen. Bei Push-Buttons hat er nur die Wahl zwischen Ja und Nein, während der Anwender bei Radio-Buttons aus einer ganzen Palette von Möglichkeiten jeweils eine auswählen kann.

Push-Buttons

Push-Buttons sind immer der Ausdruck einer Ja-/Nein-Frage, die die Applikation an den Anwender richtet. Sei es, daß die Applikation wissen möchte, ob eine Sicherungskopie der eingegebenen Daten angelegt oder nach Möglichkeit der EMS-Speicher in die Speicherverwaltung einbezogen werden soll; überall da, wo eine Information ermittelt werden muß, die intern durch die Zustände TRUE und FALSE dargestellt werden kann, kommen Push-Buttons zum Einsatz.

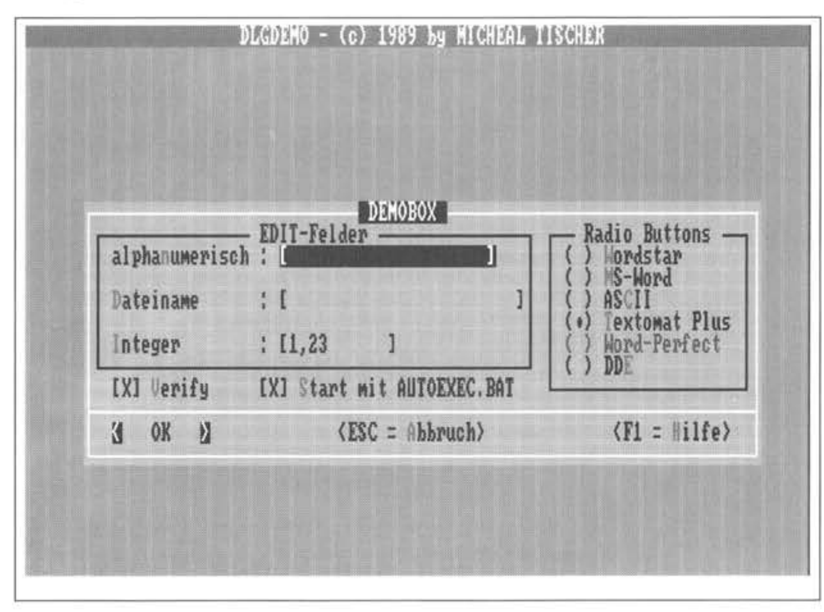


Bild 1: So präsentiert sich das Demo-Programm DLGDEMO.C. Neben drei Edit-Feldern und Action-Buttons enthält es zwei Push-Buttons und eine Gruppe von sieben Radio-Buttons.

Ihren Namen verdanken diese Objekte den An-/Ausschaltern, wie man sie oft an elektrischen Geräten findet. Drückt man den Schalter herunter, schließt sich der Stromkreis, das Gerät beginnt zu arbeiten. Zwar bleibt der Schalter in seiner Stellung, doch drückt man ihn noch ein wenig tiefer, kommt er wieder hervor und unterbricht dadurch den Stromkreis. Die beiden genannten physikalischen Zustände korrespondieren in Bezug auf einen Push-Button mit den logischen Zuständen TRUE und FALSE.

Auf dem Bildschirm wird ein Push-Button durch zwei eckige Klammern dargestellt, die entweder ein Leerzeichen oder ein großes X umschließen. Das X steht dabei für die Schalterstel-

▶ *Listing 1: DLG.H*

```
/*
[ ]──────────────────────────────────────────────────────────────────[ ]
 |                                                                     |
 |    Include-Datei    : DLG.H                                         |
 |    zur Einbindung   : der Funktionen zur Erstellung und Verwaltung von |
 |                       SAA-Dialogboxen                               |
 |    erstellt am      : 15.06.1989                                    |
 |    letztes Update am: 20.06.1989                                    |
 |    (Copyright)      : 1989 by MICHAEL TISCHER                       |
 |                                                                     |
[ ]──────────────────────────────────────────────────────────────────[ ]
*/

#include "vio.h"              /* diese beiden SAA-Include-Dateien aufnehmen */
#include "kbm.h"

/*── Makros ───────────────────────────────────────────────────────────────*/
#define F(farbe) ( dlgcol.farbe )  /* liefert Farbe aus der DLGCOL-Struktur */

/*── die folgenden Makros helfen bei der Anlage von DLGDATEN-Strukturen für */
/*── die einzelnen Objekte innerhalb einer Dialogbox                       */
#define EDIT( p )      { &p, &std_edit_fkt }
#define ACTBUT( p )    { &p, &std_ab_fkt   }
#define PUSHBUT( p )   { &p, &std_pb_fkt   }
#define RADIOBUT( p )  { &p, &std_rb_fkt   }

/*── Konstanten ───────────────────────────────────────────────────────────*/
#define HOTKEY     '#'                 /* Zeichen, das einen Hotkey markiert */
#define NOKEY      ((TASTE) 0)                              /* keine Taste */
#define NOPAINTFKT ((PAINTFKT) 0)                  /* keine 'Mal'-Funktion */

/*── Rückgabecodes für die TASTFKT ────────────────────────────────────────*/

#define TF_MAUS       250 /* aktiviere mich aufgrund eines Maus-Ereignisses */
#define TF_AKTIV      251 /* aktiviere mich aufgrund einer Taste            */
#define TF_WEITER     252 /* Taste/Mausereignis wurde nicht verarbeitet     */
#define TF_ACCEPTED   253 /* Taste/Mausereignis wurde verarbeitet           */
#define TF_FELD_VOR   254 /* ein Feld weiterschalten                        */
#define TF_FELD_RUECK 255 /* ein Feld zurück                                */
                          /* alle weiteren Codes werden als Terminations-   */
                          /* codes aufgefaßt                                */

/*── Typdeklarationen ─────────────────────────────────────────────────────*/
typedef BYTE BOOL;

typedef struct   /* Farben, die innerhalb einer Dialogbox Verwendung finden */
 {
  BYTE dlgbox,              /* Füll-Farbe für gesamte Dialogbox und Rahmen */
       nm,                                                /* normale Farbe */
       hi,                                         /* hervorgehobene Farbe */
       hk,                                         /* Farbe für den Hotkey */
       da,                           /* Farbe für inaktive Felder (disabled) */
       dk;                      /* Farbe für den Hotkey in inaktiven Feldern */
 } DLGCOL;

extern DLGCOL dlgcol;                   /* Variable mit Farben für Dialog-Boxen */

/*── Deklaration der Funktionen, die jedes Dialog-Feld bereitstellen muß ───
 |                                                                      |
 |    STARTFKT  : Wird beim Aufbau der Maske aufgerufen. Liefert dem Aufru- |
 |                fer einen Zeiger zurück, der bei allen kommenden Aufrufen |
 |                der verschiedenen Dialogfunktionen übergeben wird.      |
 |                                                                      |
 |    Jeder der folgenden Funktionen wird bei ihrem Aufruf der Zeiger über- |
 |    geben, den die STARTFKT zurückgeliefert hat und an der die Funktion |
 |    feststellen kann, welches der möglicherweise mehreren Dialog-Felder |
 |    dieses Typs angesprochen wird.                                    |
 |      AKTIVFKT  : Aktiviert ein Dialog-Feld, nachdem es zuvor über die |
 |                  Funktion CANAKTFKT seine Bereitschaft erklärt hat,  |
 |                  aktiviert zu werden. Teilt dem Dialog-Feld anhand eines |
 |                  der TF_...-Konstanten mit, warum es aktiviert wurde. |
 |      TASTFKT   : Der Aufruf erfolgt sowohl während das Dialog-Feld aktiv |
 |                  ist, manchmal aber auch, wenn ein anderes Dialog-Feld |
 |                  aktiv ist, dieses die Taste aber abgelegt hat. In diesem |
 |                  Fall muß das Dialog-Feld feststellen, ob es die Taste |
 |                  verarbeiten kann - ggf. sogar aktiviert wird.       |
 |      MAUSFKT   : Wie TASTFKT wird diese Funktion sowohl aufgerufen, wäh- |
 |                  rend ein Dialogfeld aktiv ist, als auch, wenn es nicht |
 |                  aktiv ist, um festzustellen, ob es durch ein Mausereig- |
 |                  nis aktiviert werden muß.                           |
 |      CANAKTFKT : Teilt dem Aufrufer mit, ob das Dialog-Feld bereits ist, |
 |                  aktiviert zu werden.                                |
 |      NEWVALFKT : Wird nicht vom Scheduler, sondern von einer Verbindungs- |
 |                  prozedur des Anwenders/Programmierers aufgerufen, um dem |
 |                  Dialog-Feld mitzuteilen, daß sich sein Inhalt durch ein |
 |                  äußeres Ereignis verändert hat. In den Standardversionen |
 |                  der verschiedenen Dialogtypen sind diese Funktionen |
 |                  Dummys.                                             |
 |      DEAKFKT   : Teilt dem Dialog-Feld mit, daß es daktiviert wurde. |
 |      ENDFKT    : Wird nach der Beendigung der Eingabe innerhalb der Dia- |
 |                  logbox aufgerufen, damit das Dialog-Feld einen Reset auf |
 |                  seine Daten durchführen kann. Vom Bildschirm muß es sich |
 |                  nicht entfernen.                                    |
 |                                                                      |
 ───────────────────────────────────────────────────────────────────────*/

typedef void * (*STARTFKT) ( void * iptr );
typedef void   (*NEWVALFKT)( void * iptr, void * dptr );
typedef void   (*ENDEFKT)  ( void * iptr );
typedef void   (*DEAKFKT)  ( void * iptr );
typedef void   (*AKTIVFKT) ( void * iptr, BYTE why );
typedef BYTE   (*TASTFKT)  ( void * iptr, TASTE key );
typedef BYTE   (*MAUSFKT)  ( void * iptr, BYTE x, BYTE y, BYTE ev );
typedef BOOL   (*CANAKTFKT)( void * iptr );

typedef struct         /* Struktur mit den verschiedenen Funktions-Pointern */
 {
  STARTFKT  start;
  NEWVALFKT newval;
  ENDEFKT   ende;
  TASTFKT   taste;
  DEAKFKT   deak;
  MAUSFKT   maus;
  AKTIVFKT  aktiv;
  CANAKTFKT can;
 } DLGFKT;

typedef DLGFKT *DLGFKTPTR;      /* Zeiger auf eine Dialog-Funktions-Struktur */

/*── Markos, die bei der Definition von Eingabemengen für EDIT-Felder helfen*/
#define SM(m, x, y )       EdSetMenge(m, x, y, TRUE )
#define CM(m, x, y )       EdSetMenge(m, x, y, FALSE)
#define ClearMenge(m)      CM( m, 0, 255 )
#define SetMengeAll(m)     SM( m, 0, 255 )
#define SetMengeInt(m)     ( SM( m, '0', '9' ), SM( m, '+', '-' ) )

#define SetMengeAN(m)      ( SM( m, ' ', '~' ), SM( m, 'ü', 'ü' ),  \
                             SM( m, 'ä', 'ä' ), SM( m, 'ö', 'ö' ),  \
                             SM( m, 'Ü', 'Ü' ), SM( m, 'Ö', 'Ö' ),  \
                             SM( m, 'Ä', 'Ä' ), SM( m, 'ß', 'ß' ) )
```

lung »An« (Ja), das Leerzeichen für »Aus« (Nein). Rechts neben den eckigen Klammern steht ein Text, der auf die Bedeutung des Push-Buttons hinweist und als Kurzform einer Frage betrachtet werden kann, die mit »Ja« oder »Nein« beantwortet werden muß.

Daß ein Radio-Button das aktive Dialogobjekt ist, erkennt man anhand des blinkenden Bildschirm-Cursors, der in diesem Fall zwischen den beiden eckigen Klammern erscheint. In dieser Situation kann der Button mittels verschiedener Cursor-Tasten auf TRUE oder FALSE geschaltet werden. Die Cursor-Tasten [↑] und [←] setzen den Button grundsätzlich auf TRUE, so daß zwischen den beiden Klammern das große X erscheint. Andersherum schalten die Cursor-Tasten [↓] und [→] den Button wieder in die Stellung FALSE um, so daß ein Leerzeichen das X ersetzt.

Eine Umschaltung zwischen der aktuellen Stellung des Schalters und dem Komplement kann auf verschiedene Arten realisiert werden. Ist der Button bereits als das aktive Dialogfeld ausgewählt worden, kann zu diesem Zweck z.B. die Leertaste betätigt werden.

Darüber hinaus besteht auch die Möglichkeit, den Button durch Betätigung des entsprechenden Hotkeys umzuschalten, sofern der Push-Button über einen Hotkey verfügt und dieser innerhalb der Dialogmaske angezeigt wird. Der Push-Button wird dadurch gleichzeitig zum aktuellen Dialogfeld gewählt.

Diesem Mechanismus folgt auch die Maus, die den Status des Push-Button umschaltet und ihn gleichzeitig als das aktuelle Dialogfeld auswählt, sobald sich der Mauscursor zwischen der linken eckigen Klammer und dem Ende des Button-Textes befindet und der linke Mausknopf niedergedrückt wird.

Push-Buttons sind immer Einzelgänger; auch wenn eine Dialogmaske mehrere dieser Objekte enthält, diese sogar unter- oder nebeneinander angeordnet werden, handelt es sich dabei immer um individuelle Instanzen des Dialogtyps »Push-Button«. Mit den anderen Push-Buttons innerhalb der Maske steht ein solches Objekt in keinerlei Kontakt.

Radio-Buttons

Radio-Buttons tauchen im Gegensatz zu Push-Buttons immer in »Rudeln« auf, von denen einer und nur einer zur gleichen Zeit aktiv ist. Der Anwender erhält dadurch die Möglichkeit, aus mehreren, fest vorgegebenen, Wahlmöglichkeiten die eine zu wählen, die ihm zusagt. Radio-Buttons lassen dem Anwender dadurch schon mehr Auswahlmöglichkeiten als Push-Buttons, aber immer noch weniger Freiheit als beispielsweise Edit-Felder, in die der Anwender eintragen kann, was er möchte.

Zum Einsatz kommen Radio-Buttons z.B. bei

der Auswahl eines Ausgabegeräts (COM, LPT, AUX etc.) oder bei der Einstellung der Baud-Rate für die serielle Schnittstelle (110, 150, 300, ...), doch sind dies nur zwei von unzähligen Möglichkeiten für den Einsatz dieser Art von Dialogobjekten.

Ähnlich den Push-Buttons gehen den verschiedenen Auswahl-Texten bei Radio-Buttons ebenfalls Klammern voran. Zur optischen Unterscheidung von Push-Buttons handelt es sich dabei jedoch nicht um eckige, sondern um runde Klammern, die im Fall der Auswahl auch nicht ein X, sondern einen Punkt »•« enthalten. Aus dem Text, der sich an diese Klammern anschließt, muß jeweils deutlich die Wahlmöglichkeit hervorgehen, für die der jeweilige Radio-Button steht.

Die einzelnen Auswahlmöglichkeiten, die zu einer Gruppe von Radio-Buttons gehören, werden grundsätzlich in einer Zeile nebeneinander oder in mehreren Zeilen untereinander dargestellt, wobei die Button-Klammern dann jeweils linksbündig abschließen müssen. Welche der beiden Möglichkeiten Sie als Programmierer wählen, steht Ihnen völlig frei und ist an keine Konvention gebunden. Sobald die Anzahl der Wahlmöglichkeiten jedoch eine bestimmte Größe überschreitet, ist es aus Gründen der Übersichtlichkeit ratsam, die einzelnen Buttons untereinander anzuordnen, weil das Dialogfenster sonst schnell die gesamte Bildschirmbreite einnehmen muß, was optisch nicht immer den besten Eindruck macht.

Die Auswahlmöglichkeiten zwischen den verschiedenen Buttons, die zu einer Gruppe von Radio-Buttons gehören, ähneln denen bei Push-Buttons.

Ist ein Objekt vom Typ Radio-Button als aktuelles Objekt ausgewählt worden, kann zwischen den verschiedenen Wahlmöglichkeiten mit Hilfe der Cursor-Tasten ausgewählt werden. Die Cursor-Tasten [↑] und [←] bewegen den aktiven Button (erkennbar am Punkt zwischen den runden Klammern) um eine Auswahl in Richtung auf den ersten Button, während die Cursor-Tasten [↓] und [→] ihn in die entgegengesetzte Richtung bewegen. Sind einzelne Buttons nicht wählbar (man erkennt sie an der etwas dunkleren Farbe), werden sie übersprungen.

Darüber hinaus kann jeder Radio-Button mit einem eigenem Hotkey ausgestattet werden, der am Bildschirm farblich hervorgehoben wird und durch dessen Betätigung der Button als der aktive Radio-Button innerhalb seiner Gruppe ausgewählt wird. Ist die Gruppe der Radio-Buttons zum Zeitpunkt der Betätigung des Hotkeys noch nicht das aktuelle Dialogfeld, wird sie dazu gemacht.

Ähnlich geschieht dies bei der Auswahl eines Buttons mit Hilfe der Maus. Wie auch bei Push-Buttons muß der Mauscursor nicht unbedingt zwischen die beiden runden Klammern geführt

◄ Listing 1: (Ende)

```
#define SetMengeDatei(m)  ( SM( m, 'a', 'z' ), SM( m, 'A', 'Z' ),   \
                            SM( m, '!', '%' ), CM( m, 34, 34 ),     \
                            SM( m, '-', '-' ), SM( m, '.', '.' ),   \
                            SM( m, 'ä', 'ä' ), SM( m, 'ö', 'ö' ),   \
                            SM( m, 'Ü', 'Ü' ), SM( m, 'Ö', 'Ö' ),   \
                            SM( m, 'Ä', 'Ä' ), SM( m, 'ß', 'ß' ) )

#define SetMengePath(m)   ( SetMengeDatei(m), \
                            SM( m, ':', ':' ), SM( m, 92, 92 ) )

/*— Beschreibung der Struktur eines Dialogbox-Beschreibers ————————————*/
extern DLGFKT std_edit_fkt;          /* Funktionen zur Verwaltung von Edit- */
extern DLGFKT std_ab_fkt;            /* Feldern, Action-, Push- und Radio-  */
extern DLGFKT std_pb_fkt;            /* Buttons                             */
extern DLGFKT std_rb_fkt;

typedef void (*PAINTFKT)( void );    /* Fkt. zur Gestaltung einer Dialogbox */

typedef struct           /* Daten, die für jedes Dialog-Feld benötigt werden */
 {
  void      * daten;                                /* Zeiger auf Datenblock */
  DLGFKTPTR   fkts;          /* Zeiger auf Struktur mit Dialog-Funktionen */
 } DLGDATEN;

typedef DLGDATEN * DLGDPTR;                       /* Zeiger auf Dialog-Daten */

typedef struct                                     /* beschreibt eine Dialogbox */
 {
  BYTE     x_start,                        /* Koordinaten obere linke Ecke */
           y_start,                        /* der Dialogbox                */
           x_len,                             /* Größe der Dialogbox in */
           y_len,                             /* Spalten und Zeilen     */
           anz;                               /* Anzahl der Dialogfelder */
  PAINTFKT paintfkt;                  /* Funktion zur Gestaltung der Dialogbox */
  DLGDPTR  datptr;                         /* Zeiger auf Vektor mit Zeigern */
                                           /* auf Dialogfeld-Daten          */
 } DIGDES;

typedef DIGDES * DIGDESPTR;                 /* Zeiger auf Dialogbox-Struktur */

/*— Beschreibung der Strukturen, die für ein alphanumerisches Eingabefeld -*/
/*— vom Type EDIT benötigt werden                                         -*/

typedef BYTE EMENGE[ 64 ];                                    /* Eingabemenge */
typedef BYTE *EMP;                             /* Zeiger auf eine Eingabemenge */

typedef struct                   /* beschreibt ein alphanumerisches Eingabefeld */
 {
  BOOL   enabled;                         /* darf das Feld angewählt werden? */
  BYTE   x,                          /* Anfangskoordinate relativ zur oberen */
         y,                          /* linken Ecke des Dialogbox            */
         len,                          /* Anzahl der Zeichen im Eingabefeld */
         visi;                            /* Anzahl der sichtbaren Zeichen */
  char * text;                                   /* Text vor dem Eingabefeld */
  BYTE * eingabe;                            /* Zeiger auf den Eingabepuffer */
  EMP    mptr;                     /* Zeiger auf die Menge erlaubter Zeichen */
 } EDITFELD;

/*— Beschreibung der Strukturen, die für die Verwaltung von Action Buttons */
/*— (AB) benötigt werden.                                                  */
/*— ACHTUNG! Action Buttons darf es im Gegensatz zu EDIT-Feldern, Push-    */
/*— Buttons etc. nur einmal pro Dialogbox geben                            */

typedef struct                                 /* beschreibt einen Action-Button */
 {
  BOOL   enabled;                           /* darf der AB ausgewählt werden? */
  BYTE   x,                           /* Bildschirmposition relativ zur oberen */
         y;                           /* linken Ecke des Dialogfensters        */
  char * name;                                  /* Pointer auf String mit Text */
  TASTE  key;             /* Taste, mit der dieser AP (zusätzlich zum evtl. */
                          /* vorhandenen Hotkey) verbunden wird             */
 } ONEAB;

typedef ONEAB *ABPTR;                          /* Zeiger auf einen AB-Beschreiber */

typedef struct                       /* beschreibt eine Gruppe von Action-Buttons */
 {
  BYTE anz,                                        /* Anzahl der Action-Buttons */
       standard;                                     /* Default Action-Button */
  ABPTR abp;                               /* Zeiger auf Vektor mi AB-Beschreibern */
 } ABGROUP;

/*— Beschreibung der Strukturen, die für die Verwaltung von Radio-Buttons -*/
/*— (RB) benötigt werden.                                                  */

typedef struct                                  /* beschreibt einen Radio-Button */
 {
  BOOL   enabled;                            /* darf der RB ausgewählt werden? */
  BYTE   x,                          /* Bildschirmposition relativ zur oberen */
         y;                          /* linken Ecke des Dialogfensters        */
  char * name;                                 /* Pointer auf String mit Text */
 } ONERB;

typedef ONERB *RBPTR;                          /* Zeiger auf einen RB-Beschreiber */

typedef struct    /* beschreibt eine Gruppe zusammengehöriger Radio-Buttons */
 {
  BYTE    anzahl,                                   /* Anzahl der Radio-Buttons */
        * varptr;    /* Zeiger auf Variable, die die Nummer des RB aufnimmt */
  RBPTR   rbvek;            /* Zeiger auf den Vektor mit den RB-Beschreibern */
 } RBGROUP;

typedef RBGROUP * RBGROUPTR;

/*— Beschreibung der Struktur, die für die Verwaltung von Push-Buttons ——*/
/*— (PB) benötigt wird.                                                    */
/*— ACHTUNG! Im Gegensatz zu den Radio-Buttons besteht ein Push-Button     */
/*— immer nur aus einem Eintrag, auch wenn mehrere Push-Buttons auf dem    */
/*— Bildschirm untereinander angeordnet werden                             */

typedef struct                                   /* beschreibt einen Push-Button */
 {
  BOOL   enabled;                            /* darf der PB ausgewählt werden? */
  BYTE   x,                          /* Bildschirmposition relativ zur oberen */
         y;                          /* linken Ecke des Dialogfensters        */
  char * name;                                 /* Pointer auf String mit Text */
  BOOL * varptr;   /* Zeiger auf die Variable, mit der der PB verbunden ist */
 } ONEPB;

typedef ONEPB *PBPTR;                          /* Zeiger auf einen PB-Beschreiber */

/*— öffentliche Funktionen ————————————————————————————————————————————*/

BYTE  DlgStart  ( DIGDESPTR dptr );
TASTE DlgPrint  ( BYTE x, BYTE y, char * str, BYTE fn, BYTE fh );
void  DlgDelay  ( int pauslen );
void  DlgBox    ( BYTE x, BYTE y, BYTE xlen, BYTE ylen,
                  BYTE typ, char * titel, BOOL aktiv );
void  EdSetMenge( EMP mptr, BYTE von, BYTE bis, BOOL set );

/*— Funktionsdeklarationen über Makros ————————————————————————————————*/
#define MausPause()  DlgDelay( 1 )              /* Pause bei Mausereignissen */
```

► *Listing 2: DLGPB.C*

```
/****************************************************************************/
/*                              D L G P B . C                               */
/*------------------------------------------------------------------------*/
/*   Aufgabe        : Enthält die Funktionen zur Verwaltung von Push-       */
/*                    Buttons, die innerhalb von Dialog-Boxen zum Einsatz   */
/*                    kommen.                                               */
/*                    Dieses Modul muß in Verbindumg mit dem DLG-Modul ein-*/
/*                    gesetzt werden.                                       */
/*------------------------------------------------------------------------*/
/*   Autor          : MICHAEL TISCHER                                       */
/*   entwickelt am  : 16.09.1989                                            */
/*   letztes Update : 18.09.1989                                            */
/*------------------------------------------------------------------------*/
/*   Erstellung     : CL /A[S|M|C|L|H] DLGPB.C /C                           */
/*                    dann mit dem DLG-Modul und anderen Moduln verbinden   */
/****************************************************************************/

/*— Include-Dateien einbinden ————————————————————————————————————————————*/
#include <malloc.h>
#include <string.h>
#include "dlg.h"

/*— interne Datenstruktur des Dialogtyps PB (Push Button) ————————————————*/
typedef struct /* hier werden die Daten für jeden Push-Button festgehalten -*/
 {
  BOOL    aktiv,                          /* ist der Push-Button gerade aktiv? */
          enabled,                    /* darf der Push-Button ausgewählt werden? */
        * varptr;                         /* Zeiger auf die BOOL-Variable, mit */
                                          /* der der PB verbunden ist          */
  TASTE   hotkey;             /* Hotkey des Push-Buttons (NOKEY = kein Hotkey) */
  BYTE    x1,                                      /* Startspalte des Textes */
          x2,                                        /* Endspalte des Textes */
          y;                                                        /* Zeile */
 } PBINTERN;

typedef PBINTERN *PBIP;                   /* Zeiger auf eine interne PB-Struktur */

/*— Prototypen für Funktionen, die in diesem Modul deklariert und in der —*/
/*— globalen Variablen std_pb_fkt zusammengefaßt werden                   —*/
void * pb_start ( PBPTR dptr );
void   pb_newval( PBIP pbip, void * dptr );
void   pb_ende  ( PBIP pbip );
BYTE   pb_taste ( PBIP pbip, TASTE key );
void   pb_deak  ( PBIP pbip );
void   pb_aktiv ( PBIP pbip, BYTE why );
BOOL   pb_maus  ( PBIP pbip, BYTE x, BYTE y, BYTE ev );
BOOL   pb_can   ( PBIP pbip );

/*— globale Variablen, öffentlich ————————————————————————————————————————*/
DLGFKT std_pb_fkt =                          /* Standard Dialog-Funktionen für */
 {                                           /* jeweils einen Push-Button      */
   pb_start, pb_newval, pb_ende , pb_taste,
   pb_deak,  pb_maus,   pb_aktiv, pb_can
 };

/*— globale Variablen, modulintern ———————————————————————————————————————*/
static BOOL justrel;     /* zeigt an, ober Button gerade umgeschaltet wurde */

/****************************************************************************
* PB              Es folgen die verschiedenen Funktionen des Dialogtyps PB  *
*                 ( jeweils ein Push-Button )                               *
****************************************************************************/

/****************************************************************************
*  Funktion          : p b i _ t o g g l e                                  *
**________________________________________________________________________**
*  Aufgabe           : Schaltet den Push-Button um.                         *
*  Eingabe-Parameter: PBIP = Zeiger auf den Datenblock des Push-Buttons     *
*  Return-Wert       : keiner                                               *
****************************************************************************/

void pbi_toggle( PBIP pbip )
{
 *pbip->varptr = !*pbip->varptr;                          /* Flag umdrehen */
 MouHideMouse();                                  /* Maus-Cursor ausblenden */
 VioPrint( pbip->x1+1, pbip->y, F(nm), FALSE, ( *pbip->varptr ) ? "X" : " " );
 MouShowMouse();                            /* Maus-Cursor wieder anzeigen */
}

/****************************************************************************
*  Funktion          : p b _ s t a r t                                      *
**________________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler während der Initialisierung einer *
*                      Dialogbox für jeden Push-Button aufgerufen.          *
*  Eingabe-Parameter: DPTR = Zeiger auf den Datenblock des Push-Buttons     *
*  Return-Wert       : Ein Zeiger, den der Scheduler bei allen folgenden Auf- *
*                      rufen den Funktionen pb_aktiv, pb_tast etc. übergibt. *
****************************************************************************/

void * pb_start ( PBPTR dptr )
{
 BYTE   f;                                                  /* Ausgabefarbe */
 PBIP   pbip;            /* Zeiger auf allokierte Struktur mit internen Infos */

 pbip = (PBIP) malloc( sizeof(PBINTERN) );
 pbip->varptr = dptr->varptr;                 /* Zeiger auf BOOL-Variable laden */
 pbip->enabled = dptr->enabled;                         /* Enabled-Flag merken */
 pbip->aktiv = FALSE;                            /* Push-Button ist nicht aktiv */
 pbip->x2 = VL( dptr->x + strlen( dptr->name ) + 3 );
 pbip->hotkey = DlgPrint( (pbip->x1 = VL(dptr->x)) + 4 ,
                          pbip->y = VO(dptr->y),
                          dptr->name,
                          f = (pbip->enabled = dptr->enabled)? F(nm) : F(da),
                          dptr->enabled? F(hk) : F(dk)                  );
 if ( pbip->hotkey != NOKEY )                          /* gibt es einen Hotkey? */
  --pbip->x2;                                   /* Ja, Endspalte dekrementieren */

 if ( !pbip->enabled )                                   /* ist der PB disabled? */
  pbip->hotkey = NOKEY;                              /* Ja, es gibt keinen Hotkey */

 /*— den eigentlichen Button ausgeben ———————————————————————————————————*/
 VioPrint( pbip->x1, pbip->y, f, FALSE, *(pbip->varptr) ? "[X]" : "[ ]" );
 return pbip;                               /* Zeiger an Scheduler zurückliefern */
}

/****************************************************************************
*  Funktion          : p b _ n e w v a l                                    *
**________________________________________________________________________**
*  Aufgabe           : Wird nicht direkt vom Scheduler, sondern von einer mit *
*                      ihm zusammenarbeitenden Interaktions-Funktion aufge-   *
*                      rufen, wenn sich der Inhalt der Variablen, mit der der *
*                      PB verbunden ist, durch ein äußeres Ereignis verändert *
*                      hat.                                                 *
*  Eingabe-Parameter: PBIP = der Zeiger, der beim Aufruf von pb_start zurück-*
*                             geliefert wurde.                              *
*                      DPTR = Zeiger auf eine objektspezifische Information  *
*  Return-Wert       : keiner                                               *
****************************************************************************/

void pb_newval( PBIP pbip, void * dptr )
{
}
```

werden, sondern es genügt, ihn zwischen der linken Klammer und dem Ende des Button-Textes zu positionieren. Wird dann der linke Mausknopf niedergedrückt, wird die Auswahl unter dem Mauscursor zur aktuellen Auswahl erhoben, sofern dies möglich ist. Gleichzeitig wird die Gruppe der Radio-Buttons dadurch zum aktuellen Dialogfeld, falls das nicht bereits der Fall ist.

Daß eine Gruppe von Radio-Buttons das aktuelle Dialogfeld darstellt, erkennt man übrigens immer an der Position des blinkenden Cursors, der in diesem Fall auf der Bildschirmposition erscheint, an der auch der Punkt anzutreffen ist, der die aktuelle Auswahl markiert.

Realisation der Objekte

Die Konzeption des Dialogmanagers, wie sie in der vorangegangenen Ausgabe des *Microsoft System Journals* vorgestellt wurde, kommt der Anbindung immer neuer Objekttypen entgegen, da der Dialogmanager mit diesen Objekten nur über insgesamt sieben Funktionen kommuniziert, deren Adressen ihm im Rahmen der Dialogdaten übergeben werden. So fällt es auch nicht schwer, die bisherigen Dialogobjekte um die beiden neuen Objekte »Push-Buttons« und »Radio-Buttons« zu erweitern.

Der Programmcode für die beiden Objekte findet sich in den Moduln `DLGPB.C` (Push-Buttons) und `DLGRB.C` (Radio-Buttons), deren Listing auf den folgenden Seiten abgedruckt ist.

Die Typendeklarationen, die Sie zur Anlage von Dialogfeldern dieser Art innerhalb Ihrer Programme benötigen, finden Sie innerhalb der `INCLUDE`-Datei `DLG.H`. Sie wurde bereits in der letzten Ausgabe vorgestellt, nun jedoch um die neuen Deklarationen für Push-Buttons und Radio-Buttons erweitert. Ersetzen Sie die Datei `DLG.H` aus der letzten Ausgabe deshalb bitte durch die neue, die Sie zusammen mit den verschiedenen C-Modulen auf der Diskette im Heft finden.

Die Typdeklarationen innerhalb von `DLG.H` sind für Sie die einzige Schnittstelle zwischen Ihrem Programm und den Dialogobjekten, da Sie weder mit dem Programmcode der Objekte, noch mit deren interner Realisation in Berührung kommen. Nur wenn Sie in die Arbeitsweise der Objekte eingreifen möchten, um beispielsweise Meta-Objekte zu bilden, die aus den Grundobjekten »Push-Button« oder »Radio-Button« hervorgehen, kommen Sie mit den Interna beiden Modulen `DLGRB.C` und `DLGPB.C` in Berührung.

Der Aufbau dieser Objekte entspricht dem standardisierten Aufbau von Dialogobjekten, wie er in der vorangegangenen Folge vorgestellt wurde. Bitte konsultieren Sie deshalb das Dokumentationsmaterial aus der genannten Folge, falls Sie Fragen zu der internen Funktionsweise der Objekte haben.

Deklaration von Push-Buttons

Objekte vom Typ »Push-Button« werden durch eine Struktur mit der Typbezeichnung ONEPB definiert, deren Deklaration Teil der Include-Datei DLG.H ist. Hier wird anhand eines Flags vom Typ BOOL zunächst festgehalten, ob der Push-Button während der Eingabe innerhalb des Dialogfelds verändert werden darf oder nicht. Die Konstanten TRUE und FALSE repräsentieren die beiden Möglichkeiten.

```
typedef struct                                    /* beschreibt einen Push-Button */
 {
  BOOL   enabled;                                /* darf der PB ausgewählt werden? */
  BYTE   x,y;            /* Bildschirmposition relativ zur oberen linken Ecke */
  char * name;                                     /* Pointer auf String mit Text */
  BOOL * varptr;   /* Zeiger auf die Variable, mit der der PB verbunden ist */
 } ONEPB;
```

Die Position des Push-Buttons innerhalb der Dialogbox wird in den folgenden beiden Feldern mit den Namen X und Y festgehalten. Die beiden Variablen geben die Enfernung der linken Klammer am Anfang des Buttons von der oberen linken Ecke der Dialogbox in Spalten und Zeilen an.

Der Text, der der Klammer am Anfang des Buttons folgt, wird durch eine Variable referenziert, die auf die Variablen X und Y folgt. Ihre Bezeichnung lautet name und sie stellt einen Zeiger auf den entsprechenden String dar. Wie bereits bei Edit-Feldern und Action-Buttons können Sie dem Push-Button einen Hotkey verleihen, indem Sie dem entsprechenden Buchstaben innerhalb des Namens ein Doppelkreuz (#) voranstellen.

Als letzte Information wird innerhalb der Struktur ein Zeiger auf die Variable vom Typ BOOL verzeichnet, deren Inhalt den Status des Push-Buttons widerspiegelt. Bei der Öffnung des Dialogfelds wird der Initialstatus des Push-Buttons dieser Variable entnommen und nach dem Abschluß der Eingabe wieder in diese Variable eingetragen.

Um ein Dialogobjekt vom Typ »Push-Button« in eine Dialogmaske aufzunehmen, genügt es jedoch nicht, eine Struktur vom Typ ONEPB zu definieren; Sie müssen diese Struktur auch innerhalb des Vektors aufführen, der die einzelnen Dialogfelder und ihre Reihenfolge gegenüber dem Dialogmanager identifiziert und ihm beim Aufruf der Funktion DlgStart übergeben wird.

Wie bereits für die Dialogtypen »Edit« und »Action-Button« enthält die INCLUDE-Datei DLG.H jetzt auch ein Makro, das Sie bei der Angabe eines Objekts vom Typ »Push-Button« innerhalb des Objektvektors unterstützt. Es trägt den Namen PUSHBUT und erwartet als Argument den Namen der Variable vom Typ ONEPB, in der Sie die Objektdaten abgelegt haben.

Aufgabe dieses Makros ist es, Sie von der Angabe der Dialogfunktionen zu isolieren, so daß Sie mit ihnen gar nicht in Kontakt kommen. So lange Sie dieses Makro einsetzen, wird der Dia-

Listing 2: (Fortsetzung)

```
/*****************************************************************************
*  Funktion         : p b _ e n d e                                          *
**___________________________________________________________________________**
*  Aufgabe          : Wird vom Scheduler während des Schließens der Dialog-  *
*                     box aufgerufen, um PB Gelgegenheit zu geben, Clean-Up  *
*                     Arbeiten durchzuführen.                                *
*  Eingabe-Parameter: PBIP = der Zeiger, der beim Aufruf von pb_start zurück-*
*                            geliefert wurde.                                *
*  Return-Wert      : keiner                                                 *
*****************************************************************************/
void pb_ende( PBIP pbip )
{
 free( pbip );                     /* die allokierte Struktur wieder freigeben */
}

/*****************************************************************************
*  Funktion         : p b _ t a s t e                                        *
**___________________________________________________________________________**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um dem Dialog-Feld eine *
*                     Taste zu übergeben.                                    *
*  Eingabe-Parameter: PBIP  = der Zeiger, der beim Aufruf von pb_start zu-   *
*                             rückgeliefert wurde.                           *
*                     TASTE = Code der zu bearbeitenden Taste                *
*  Return-Wert      : Reaktionscode (TF_...)                                 *
*****************************************************************************/
BYTE pb_taste( PBIP pbip, TASTE key )
{
 if ( pbip->aktiv )                              /* ist der Push-Button aktiv? */
  {                                                                   /* Ja */
   switch ( key )                                           /* Taste auswerten */
    {
     case CUP    : /*------------- Cursor-Up und -Left schalten Button an -*/
     case CLEFT  :
       *pbip->varptr = FALSE;                 /* BOOL-Variable auf TRUE setzen */
       pbi_toggle( pbip );                    /* (das erledigt pbi_toggle() )  */
       return TF_ACCEPTED;                           /* Taste wurde angenommen */

     case CDOWN  : /*---------- Cursor-Down und -Right schalten Button aus -*/
     case CRIGHT :
       *pbip->varptr = TRUE;                 /* BOOL-Variable auf FALSE setzen */
       pbi_toggle( pbip );                   /* (das erledigt pbi_toggle() )   */
       return TF_ACCEPTED;                           /* Taste wurde angenommen */

     case ' '    : /*------------- die SPACE-Taste schaltet den Button um -*/
       pbi_toggle( pbip );                                   /* Flag umdrehen */
       return TF_ACCEPTED;                           /* Taste wurde angenommen */

     case TAB    : /*------------------------ TAB springt zum nächsten Feld -*/
       return TF_FELD_VOR;

     case BACKTAB: /*---------- SHIFT+TAB springt zum vorhergehenden Feld -*/
       return TF_FELD_RUECK;

     default     : /*------------- jede andere Taste, ist es der Hotkey? -*/
       if ( key == pbip->hotkey )                                 /* Hotkey? */
        {                                                             /* Ja */
         pbi_toggle( pbip );                                 /* Flag umdrehen */
         return TF_ACCEPTED;
        }
       else                                         /* Nein, nicht der Hotkey */
        return TF_WEITER;
    }
  }
 else           /* der Push-Button ist noch nicht aktiv, auf Hotkey testen */
  if ( key == pbip->hotkey )                                      /* Hotkey? */
   {                                                                  /* Ja */
    pbi_toggle( pbip );                                      /* Flag umdrehen */
    return TF_AKTIV;
   }
  else                                              /* Nein, nicht der Hotkey */
   return TF_WEITER;
}

/*****************************************************************************
*  Funktion         : p b _ d e a k                                          *
**___________________________________________________________________________**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um den Push-Button von  *
*                     seiner Deaktivierung in Kenntnis zu setzen.            *
*  Eingabe-Parameter: PBIP = der Zeiger, der beim Aufruf von pb_start zurück-*
*                            geliefert wurde.                                *
*  Return-Wert      : keiner                                                 *
*****************************************************************************/
void pb_deak( PBIP pbip )
{
 pbip->aktiv = FALSE;                                   /* neuen Status merken */
 VioHideCursor();                            /* Cursor vom Bildschirm entfernen */
}

/*****************************************************************************
*  Funktion         : p b _ m a u s                                          *
**___________________________________________________________________________**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um dem Dialog-Feld ein  *
*                     Mausereignis zu übergeben                              *
*  Eingabe-Parameter: PBIP  = der Zeiger, der beim Aufruf von pb_start zu-   *
*                             rückgeliefert wurde.                           *
*                     X, Y  = Position des Mauscursors relativ zu oberen     *
*                             linken Bildschirmecke                          *
*                     EV    = Event-Maske, die das Ereignis beschreibt, um   *
*                             dessentwillen die Funktion aufgerufen wird     *
*  Return-Wert      : Reaktionscode (TF_...)                                 *
*****************************************************************************/

BYTE pb_maus( PBIP pbip, BYTE x, BYTE y, BYTE ev )
{
 if ( pbip->aktiv )                             /* ist der Push-Buttons aktiv? */
  {                                                                   /* Ja */
   if ( ev & EV_LEFT_REL )       /* wurde der linke Mausbutton losgelassen? */
    {              /* Ja, feststellen, ob sich die Maus über dem PB befand */
     if ( justrel == FALSE)        /* Mausknopf gerade schon umgeschaltet? */
      {                                                             /* Nein */
       justrel = TRUE;                                          /* jetzt aber */
       if ( pbip->y == y  &&  x >= pbip->x1  &&  x <= pbip->x2  )
        {                                  /* Maus über Push-Button und Button */
         if ( pbip->enabled )                       /* ist der Button enabled? */
          pbi_toggle( pbip );                            /* Ja, Flag umdrehen */
         return TF_ACCEPTED;                      /* Ereignis wurde verarbeitet */
        }
       else                                 /* die Maus befindet sich nicht über */
        return TF_WEITER;                   /* dem aktuellen Button             */
      }
     else                              /* Mausknopf wurde bereits umgeschaltet */
      return TF_ACCEPTED;                     /* Mausereignis trotzdem annehmen */
    }
   else                                    /* Mausknopf wurde nicht losgelassen */
    {
     if ( ev & EV_LEFT_PRESS )                      /* wurde Mausknopf betätigt? */
      {                                                               /* Ja */
       if ( pbip->y == y  &&  x >= pbip->x1  &&  x <= pbip->x2  )
        {                                     /* Maus über dem aktuellen Button */
         justrel = FALSE;                   /* Umschaltung kann wieder erfolgen */
         return TF_ACCEPTED;                             /* Ereignis aktzeptieren */
        }
```

▶ *Listing 2: (Ende)*

```
      else                                 /* Maus nicht über aktuellem Button */
       return TF_WEITER;                           /* Ereignis weiterreichen */
      }
    else                                    /* unbekanntes Mausereignis */
     return TF_WEITER;
   }
  }
 else                               /* Nein, der Push-Button ist nicht aktiv */
  {
   if  ( ev & EV_LEFT_PRESS ) /* wurde der linke Mausbutton niedergedrückt? */
    if ( pbip->enabled  &&  pbip->y == y  &&
         x >= pbip->x1  &&  x <= pbip->x2  )                         /* Ja */
      return TF_MAUS;           /* Maus über dem Button und Button enablded */
   return TF_WEITER;                               /* Mausereignis abweisen */
  }
}

/***************************************************************************
*  Funktion          : p b _ c a n                                         *
**_____________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler aufgerufen, um festzustellen, ob  *
*                      das Dialogfeld bereit ist, aktiviert zu werden.     *
*  Eingabe-Parameter: PBIP = der Zeiger, der beim Aufruf von pb_start zurück-*
  *                          geliefert wurde.                              *
*  Return-Wert       : TRUE, wenn das Dialog-Feld aktiviert werden kann,   *
*                      sonst FALSE                                         *
***************************************************************************/

BOOL pb_can( PBIP pbip )
{
 return pbip->enabled;                            /* Enabled-Flag entscheidet */
}

/***************************************************************************
*  Funktion          : p b _ a k t i v                                     *
**_____________________________________________________________________**
*  Aufgabe           : Wird vom Scheduler aufgerufen, um das Dialog-Feld von *
*                      seiner Aktivierung in Kenntnis zu setzen.           *
*  Eingabe-Parameter: PBIP = der Zeiger, der beim Aufruf von pb_start zurück-*
*                             geliefert wurde.                             *
*                      WHY  = warum wird das Feld aktiviert                *
*  Return-Wert       : keiner                                              *
***************************************************************************/

void pb_aktiv( PBIP pbip, BYTE why )
{
 justrel = FALSE;
 pbip->aktiv = TRUE;                                 /* neuen Status merken */
 VioSetCursor( pbip->x1+1, pbip->y );          /* Cursor auf den PB setzen */
}
```

logmanager immer die vorgegebenen Dialogfunktionen aus der Datei DLGPB.C aufrufen und sich das Objekt dadurch in der oben beschriebenen Art und Weise verhalten.

Deklaration von Radio-Buttons

Radio-Buttons werden durch eine Datenstruktur mit dem Namen RBGROUP beschrieben, die ebenfalls innerhalb der INCLUDE-Datei DLG.H deklariert wird. Sie nimmt zunächst die Anzahl der einzelnen Buttons innerhalb einer Gruppe von Radio-Buttons im Feld anzahl auf.

Darüber hinaus wird dort mit der Komponente varptr auch ein Zeiger auf die Variable vom Typ BYTE erfaßt, die die Nummer des aktuell ausgewählten Radio-Buttons aufnimmt.

```
typedef struct    /* beschreibt eine Gruppe zusammengehöriger Radio-Buttons */
 {
  BYTE    anzahl,                               /* Anzahl der Radio-Buttons */
        * varptr;   /* Zeiger auf Variable, die die Nummer des RB aufnimmt */
  RBPTR   rbvek;           /* Zeiger auf den Vektor mit den RB-Beschreibern */
 } RBGROUP;
```

Analog zu der Verfahrensweise bei Push-Buttons gibt ihr Inhalt beim Öffnen der Dialogbox die Nummer des aktiven Radio-Buttons an und nimmt nach dem Abschluß der Eingabe auch die Nummer des Radio-Buttons auf, den der Anwender während der Eingabe ausgewählt hat. Der Inhalt dieser Variable bewegt sich immer zwischen 0 (erster Radio-Button ausgewählt) und der Anzahl der Radio-Buttons minus 1 (letzter Button ausgewählt). Sie kann dadurch vom Aufrufer der Dialogbox leicht als Index in einen Vektor benutzt oder im Rahmen einer switch-Anweisung eingesetzt werden.

Als letzte Komponente wird innerhalb einer Datenstruktur vom Typ RBGROUP ein Zeiger auf einen Vektor aufgeführt, der die einzelnen Radio-Buttons genauer beschreibt. Diese Komponente trägt den Namen rbvek und verweist auf Strukturen vom Typ ONERB.

Diese Struktur besteht aus vier Komponenten, deren Inhalt den verschiedenen Komponenten innerhalb von ONEPB gleicht. Hier kann neben der Position des Buttons relativ zur oberen linken Ecke der Dialogbox auch der zugehörige Button-Text ausgewählt werden, in dem wie bei den Push-Buttons ein Hotkey aufgeführt werden kann.

Über das Feld enabled können Sie verschiedene Buttons von der Auswahl ausschließen, doch müssen Sie dafür Sorge tragen, daß immer mindestens ein Button wählbar bleibt. Bei mindestens einem Radio-Button muß dieses Feld also den Wert TRUE enthalten.

Eine Demo-Dialogbox

Wie schon in der vorangegangenen Folge dieser Serie möchte ich Ihnen auch diesmal wieder ein kleines Demoprogramm anbieten, das Ihnen die Anlage der verschiedenen Datenstukturen verdeutlicht, die der Dialogmanager und die Dialogfunktionen der verschiedenen Objekte zur Verwaltung der Objekte benötigen. DLGDEMO.C ist sein Name, und Sie finden es wie auch alle anderen Dateien auf der Diskette in diesem Heft.

Neben den Moduln DLG.C, DLGEDIT.C und DLG-AB.C, die in der vorangegangenen Folge vorgestellt wurden, benötigen Sie zur Kompilation dieses Programms auch die Module VIO.C und KBM.C aus den ersten beiden Folgen dieser Serie.

Wie Sie aus diesen Modulen ein ausführbares Programm erzeugen können, verrät Ihnen der Kommentar im Kopf der Datei DLGDEMO.C.
Michael Tischer

Michael Tischer ist freier EDV-Journalist, Programmierer und Fachbuchautor. Zu seinen bekanntesten Werken zählt das Buch »PC-Intern« aus dem Data-Becker-Verlag, das hierzulande als das Standard-Werk zur PC-Programmierung gilt. Darüber hinaus schreibt Michael Tischer für verschiedene EDV-Fachzeitschriften und ist mit seiner SAA-Serie ständiger Gast im Microsoft System Journal.

Listing 3: DLGRB.C

```c
/***************************************************************************/
/*                              D L G R B . C                              */
/*-------------------------------------------------------------------------*/
/*    Aufgabe        : Enthält die Funktionen zur Verwaltung von Radio-    */
/*                     Buttons, die innerhalb von Dialog-Boxen zum Einsatz */
/*                     kommen.                                             */
/*                     Dieses Modul muß in Verbindumg mit dem DLG-Modul ein-*/
/*                     gesetzt werden.                                     */
/*-------------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                     */
/*    entwickelt am  : 16.09.1989                                          */
/*    letztes Update : 18.09.1989                                          */
/*-------------------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] DLGRB.C /C                         */
/*                     dann mit dem DLG-Modul und anderen Moduln verbinden */
/***************************************************************************/

/*-- Include-Dateien einbinden --------------------------------------------*/

#include <malloc.h>
#include <string.h>
#include "dlg.h"

/*-- interne Datenstruktur des Dialogtyps RB (Radio Button) ---------------*/

typedef struct      /* interne Informationen für jeweils einen Radio-Button */
 {
  TASTE hotkey;                                     /* der Hotkey des RB */
  BOOL  enabled;                                          /* RB wählbar? */
  BYTE  y,                                                      /* Zeile */
        x1, x2;                           /* erste Spalte, letzte Spalte */
 } RBINFO;

typedef struct                   /* Daten für jede Gruppe von Radio-Buttons */
 {
  BOOL     aktiv,                           /* ist das Objekt gerade aktiv? */
           mouout;    /* wurde Mausknopf  außerhalb eines RB losgelassen? */
  RBINFO * infoptr;                /* Zeiger auf den Vektor mit den Hotkeys */
  BYTE     anz,               /* Anzahl der Radio-Buttons in dieser Gruppe */
         * varptr;                  /* Zeiger auf die zugehörige Variable */
 } RBINTERN;

typedef RBINTERN *RBIP;               /* Zeiger auf eine interne RB-Struktur */

/*-- Prototypen für Funktionen, die in diesem Modul deklariert und in der --*/
/*-- globalen Variablen std_rb_fkt zusammengefaßt werden                  --*/

void * rb_start ( RBGROUPTR dptr );
void   rb_newval( RBIP rbip, void * dptr );
void   rb_ende  ( RBIP rbip );
BYTE   rb_taste ( RBIP rbip, TASTE key );
void   rb_deak  ( RBIP rbip );
void   rb_aktiv ( RBIP rbip, BYTE why );
BOOL   rb_maus  ( RBIP rbip, BYTE x, BYTE y, BYTE ev );
BOOL   rb_can   ( RBIP rbip );

/*-- globale Variablen, öffentlich ----------------------------------------*/
DLGFKT std_rb_fkt =                      /* Standard Dialog-Funktionen für */
 {                                       /* jeweils einen Radio-Button     */
   rb_start, rb_newval, rb_ende , rb_taste,
   rb_deak,  rb_maus,   rb_aktiv, rb_can
 };

/*-- globale Variablen, modulintern ---------------------------------------*/
static BOOL justrel;     /* zeigt an, ober Button gerade umgeschaltet wurde */

/****************************************************************************
* RB           Es folgen die verschiedenen Funktionen des Dialogtyps RB    *
*              ( jeweils eine Gruppe von Radio-Buttons )                   *
***************************************************************************/

/****************************************************************************
*  Funktion         : r b i _ n e x t                                      *
**------------------------------------------------------------------------**
*  Aufgabe          : Interne Funktion, die zur Auswahl eines neuen RBs nach *
*                     der Betätigung einer Cursor-Taste oder einer Maus-   *
*                     auswahl aufgerufen wird.                             *
*  Eingabe-Parameter: RBIP   = Zeiger auf den internen Datenblock          *
*                     OFFSET = Suchrichtung (+1, -1 etc.)                  *
*  Return-Wert      : keiner                                               *
***************************************************************************/

void rbi_next( RBIP rbip, int offset )
{
 RBINFO * ifp;                       /* Laufzeiger in den internen RB-Vektor */
 int      i;                                       /* Nummer des neuen RB */

 i = *rbip->varptr;        /* I mit der Nummer des aktuellen Elements laden */
 ifp = rbip->infoptr+i;        /* ifp auf aktuelles Element im Vektor setzen */

 MouHideMouse();                                   /* Maus-Cursor ausblenden */
 VioPrint( ifp->x1+1, ifp->y, F(nm), FALSE, " " );     /* Button ausblenden */


 /*-- den nächsten RB in der über OFFSET angegebenen Richtung suchen, der --*/
 /*-- enabled ist                                                         --*/
 do
  {
   i += offset;             /* I mit der Nummer des nächsten Elements laden */
   if ( i < 0 )                               /* Wrap-Around zum letzten RB? */
    ifp = rbip->infoptr + (i = rbip->anz-1 );
   else                                                              /* Nein */
    if ( i == rbip->anz )                     /* Wrap-Around zum ersten RB? */
     {                                                                 /* Ja */
      ifp = rbip->infoptr;
      i = 0;
     }
    else                                             /* gar kein Wrap-Around */
     ifp += offset;
  }
 while ( ifp->enabled == FALSE );

 /*-- es wurden ein Button gefunden, der enabled ist ----------------------*/
 *rbip->varptr = i;                    /* Nummer des neuen akt. Button setzen */
 VioPrint( ifp->x1+1, ifp->y, F(nm), FALSE, "\x07" );   /* Button markieren */
 VioSetCursor( ifp->x1+1, ifp->y );                      /* Cursor draufsetzen */
 MouShowMouse();                              /* Maus-Cursor wieder einblenden */
}

/****************************************************************************
*  Funktion         : r b i _ h o t k e y                                  *
**------------------------------------------------------------------------**
*  Aufgabe          : Interne Funktion, die bei der Suche nach einem Hotkey *
*                     innerhalb eines RBs eingesetzt wird.                 *
*  Eingabe-Parameter: RBIP   = Zeiger auf den internen Datenblock          *
*                     KEY    = die betätigte Taste und vermeintlicher Hotkey *
*  Return-Wert      : TF_ACCEPTED, wenn es sich um einen Hotkey handelt und *
*                     TF_WEITER, wenn die Taste nicht als ein solcher iden- *
*                     tifiziert werden konnte.                             *
*  Info             : Wird die Taste als Hotkey erkannt, wird auch gleich  *
*                     der neue RB angezeigt.                               *
***************************************************************************/

BYTE rbi_hotkey( RBIP rbip, TASTE key )
{
 BYTE     i;                                          /* Schleifenzähler */
 RBINFO * lrip;        /* Laufzeiger in internen Vektor mit RB-Beschreibern */

 for ( lrip = rbip->infoptr, i = 0;
       i < rbip->anz  &&  lrip->hotkey != key;
       ++i, ++lrip );
 if ( i < rbip->anz )                          /* wurde der Hotkey entdeckt? */
  {                                                                    /* Ja */
   rbi_next( rbip, i-*rbip->varptr );                  /* neuen RB anzeigen */
   return TF_ACCEPTED;
  }
 else                                       /* der Hotkey wurde nicht entdeckt */
  return TF_WEITER;
}

/****************************************************************************
*  Funktion         : r b i _ m a u s                                      *
**------------------------------------------------------------------------**
*  Aufgabe          : Interne Funktion, die feststellt, ob sich die angege- *
*                     bene Bildschirmposition innerhalb eines der verschie- *
*                     denen Radio-Buttons befindet.                        *
*  Eingabe-Parameter: RBIP   = Zeiger auf den internen Datenblock          *
*                     X, Y   = absolute Bildschirmposition                 *
*  Return-Wert      : Die Nummer des RB oder -1, wenn sich kein RB an der  *
*                     angegebenen Position befindet.                       *
***************************************************************************/

int rbi_maus( RBIP rbip, BYTE x, BYTE y )
{
 BYTE     i;                                          /* Schleifenzähler */
 RBINFO * lrip;        /* Laufzeiger in internen Vektor mit RB-Beschreibern */

 for ( lrip = rbip->infoptr, i = 0; i < rbip->anz; ++i, ++lrip )
  if ( lrip->y == y  &&  x >= lrip->x1  &&  x <= lrip->x2 )
   return i;                                               /* RB entdeckt! */

 return -1;                                             /* keinen RB entdeckt */
}

/****************************************************************************
*  Funktion         : r b _ s t a r t                                      *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler während der Initialisierung einer *
*                     Dialogbox für jeden Radio-Button aufgerufen.         *
*  Eingabe-Parameter: DPTR = Zeiger auf den Datenblock des Radio-Buttons   *
*  Return-Wert      : Ein Zeiger, den der Scheduler bei allen folgenden Auf- *
*                     rufen den Funktionen rb_aktiv, rb_tast etc. übergibt. *
***************************************************************************/

void * rb_start ( RBGROUPTR dptr )
{
 BYTE    i,                                           /* Schleifenzähler */
         f;                                              /* Ausgabefarbe */
 RBPTR   lrbptr;              /* Laufzeiger in den übergebenen RB-Vektor */
 RBINFO *lrifp;                  /* Laufzeiger in den internen RB-Vektor */
 RBIP    rbip;        /* Zeiger auf allokierte Struktur mit internen Infos */

 /*-- Puffer für interne Struktur und zugehörigen Vektor allokieren -------*/

 rbip = (RBIP) malloc( sizeof(RBINTERN) );
 rbip->infoptr = (RBINFO *) malloc( dptr->anzahl * sizeof( RBINFO ) );

 rbip->anz = dptr->anzahl;                        /* Anzahl der Buttons merken */
 rbip->varptr = dptr->varptr;               /* Zeiger auf BYTE-Variable laden */
 rbip->aktiv = FALSE;                            /* das Objekt ist nicht aktiv */

 /*-- die einzelnen Radio-Buttons durchlaufen, dabei Informationen in den --*/
 /*-- Vektor eintragen und die RBs ausgeben                               --*/

 for ( lrbptr = dptr->rbvek, lrifp = rbip->infoptr, i = 0;
       i < dptr->anzahl;
       ++i, ++lrbptr, ++lrifp )
  {
   lrifp->x2 = VL( lrbptr->x + strlen( lrbptr->name ) + 3 );

   lrifp->hotkey =
     DlgPrint( (lrifp->x1 = VL(lrbptr->x)) + 4 ,
               lrifp->y = VO(lrbptr->y),
               lrbptr->name,
               f = (lrifp->enabled = lrbptr->enabled)? F(nm) : F(da),
               lrbptr->enabled? F(hk) : F(dk)                     );
   VioPrint( lrifp->x1, lrifp->y, f, FALSE,
             ( *rbip->varptr == i ) ? "(\x07)" : "( )" );

   if ( lrifp->hotkey != NOKEY )                    /* gibt es einen Hotkey? */
    --lrifp->x2;                              /* Ja, Endspalte dekrementieren */

   if ( !lrifp->enabled )                             /* ist der RB disabled? */
     lrifp->hotkey = NOKEY;                     /* Ja, es gibt keinen Hotkey */
  }

 return rbip;                             /* Zeiger an Scheduler zurückliefern */
}

/****************************************************************************
*  Funktion         : r b _ n e w v a l                                    *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird nicht direkt vom Scheduler, sondern von einer mit *
*                     ihm zusammenarbeitenden Interaktions-Funktion aufge- *
*                     rufen, wenn sich der Inhalt der Variablen, mit der der *
*                     RB verbunden ist, durch ein äußeres Ereignis verändert *
*                     hat.                                                 *
*  Eingabe-Parameter: RBIP = der Zeiger, der beim Aufruf von rb_start zurück-*
*                            geliefert wurde.                              *
*                     DPTR = Zeiger auf eine objektspezifische Information *
*  Return-Wert      : keiner                                               *
***************************************************************************/

void rb_newval( RBIP rbip, void * dptr )
{
}


/****************************************************************************
*  Funktion         : r b _ e n d e                                        *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler während des Schließens des Dialog- *
*                     box aufgerufen, um RB Gelegegenheit zu geben, Clean-Up *
*                     Arbeiten durchzuführen.                              *
*  Eingabe-Parameter: RBIP = der Zeiger, der beim Aufruf von rb_start zurück-*
*                            geliefert wurde.                              *
*  Return-Wert      : keiner                                               *
***************************************************************************/

void rb_ende( RBIP rbip )
{
 free( rbip->infoptr );      /* Vektor mit internen RB-Beschreibern freig. */
 free( rbip );                  /* die allokierte Struktur wieder freigeben */
}
```

▶ *Listing 3: (Fortsetzung)*

```c
/***************************************************************************
*  Funktion         : r b _ t a s t e                                      *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um dem Dialog-Feld eine *
*                     Taste zu übergeben.                                   *
*  Eingabe-Parameter: RBIP  = der Zeiger, der beim Aufruf von rb_start zu-  *
*                             rückgeliefert wurde.                          *
*                     TASTE = Code der zu bearbeitenden Taste               *
*  Return-Wert      : Reaktionscode (TF_...)                                *
***************************************************************************/

BYTE rb_taste( RBIP rbip, TASTE key )
{
 if ( rbip->aktiv )                                 /* ist das Objekt aktiv? */
  {                                                                /* Ja */
   switch ( key )                                      /* Taste auswerten */
    {
     case CUP    : /*---- Cursor-Up und -Left springen einen RB nach oben -*/
     case CLEFT  :
       rbi_next( rbip, -1 );                     /* nächsten Button markieren */
       return TF_ACCEPTED;                         /* Taste wurde angenommen */

     case CDOWN  : /*- Cursor-Down und -Right springen einen RB nach unten -*/
     case CRIGHT :
       rbi_next( rbip, 1 );                      /* nächsten Button markieren */
       return TF_ACCEPTED;                         /* Taste wurde angenommen */

     case TAB    : /*-------------------- TAB springt zum nächsten Feld -*/
       return TF_FELD_VOR;

     case BACKTAB: /*---------- SHIFT+TAB springt zum vorhergehenden Feld -*/
       return TF_FELD_RUECK;

     default     : /*--------------------- andere Taste, Hotkey testen -*/
       return rbi_hotkey( rbip, key );
    }
  }
 else            /* der Radio-Button ist noch nicht aktiv, auf Hotkey testen */
  return ( rbi_hotkey( rbip, key ) == TF_ACCEPTED ) ? TF_AKTIV : TF_WEITER;
}

/***************************************************************************
*  Funktion         : r b _ d e a k                                        *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um den Radio-Button von *
*                     seiner Deaktivierung in Kenntnis zu setzen.           *
*  Eingabe-Parameter: RBIP = der Zeiger, der beim Aufruf von rb_start zurück-*
*                            geliefert wurde.                               *
*  Return-Wert      : keiner                                                *
***************************************************************************/

void rb_deak( RBIP rbip )
{
 rbip->aktiv = FALSE;                                   /* neuen Status merken */
 VioHideCursor();                              /* Cursor vom Bildschirm entfernen */
}

/***************************************************************************
*  Funktion         : r b _ m a u s                                        *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um dem Dialog-Feld ein *
*                     Mausereignis zu übergeben                             *
*  Eingabe-Parameter: RBIP  = der Zeiger, der beim Aufruf von rb_start zu-  *
*                             rückgeliefert wurde.                          *
*                     X, Y  = Position des Mauscursors relativ zu oberen    *
*                             linken Bildschirmecke                         *
*                     EV    = Event-Maske, die das Ereignis beschreibt, um  *
*                             dessentwillen die Funktion aufgerufen wird    *
*  Return-Wert      : Reaktionscode (TF_...)                                *
***************************************************************************/

BYTE rb_maus( RBIP rbip, BYTE x, BYTE y, BYTE ev )
{
 int mourb;                               /* für Suche nach einem Radio-Button */

 if ( rbip->aktiv  &&  !rbip->mouout )      /* RB aktiv und nicht außerhalb? */
  {                                                                /* Ja */
   if ( ev & EV_LEFT_REL )        /* wurde der linke Mausbutton losgelassen? */
    {               /* Ja, feststellen, ob sich die Maus über dem RB befand */
     if ( ( mourb = rbi_maus( rbip, x, y ) ) != -1 )
      {                                          /* Maus über einem Radio-Button */
       if ( justrel == FALSE)        /* Mausknopf gerade schon umgeschaltet? */
        {                                                          /* Nein */
         justrel = TRUE;                                       /* jetzt aber */
         if ( (rbip->infoptr+mourb)->enabled )        /* ist der RB enabled? */
          rbi_next( rbip, mourb-*rbip->varptr );             /* Ja, anwählen */
         else                             /* Nein, Cursor wieder auf akt. RB */
          VioSetCursor( (rbip->infoptr+*rbip->varptr)->x1+1,
                        (rbip->infoptr+*rbip->varptr)->y );
         return TF_ACCEPTED;                       /* Ereignis wurde verarbeitet */
        }
       else                               /* Mausknopf wurde bereits umgeschaltet */
        return TF_ACCEPTED;                     /* Mausereignis trotzdem annehmen */
      }
     else                         /* die Maus befindet sich nicht über einem RB */
      {
       VioSetCursor( (rbip->infoptr+*rbip->varptr)->x1+1,     /* Cursor auf */
                     (rbip->infoptr+*rbip->varptr)->y );   /* akt RB     */
       justrel = TRUE;                                     /* gerade losgelassen */
       rbip->mouout = TRUE;                 /* außerhalb der Buttons losgelassen */
       return TF_WEITER;                          /* Ereignis nicht akzeptieren */
      }
    }
   else                                   /* Mausknopf wurde nicht losgelassen */
    {
     if ( ev & EV_LEFT_PRESS )                      /* wurde Mausknopf betätigt? */
      {                                                                /* Ja */
       if ( ( mourb = rbi_maus( rbip, x, y ) ) != -1 )
        {                                   /* Maus über einem der Radio-Buttons */
         justrel = FALSE;                   /* Umschaltung kann wieder erfolgen */
         if ( (rbip->infoptr+mourb)->enabled )    /* ist der Button enabled? */
          VioSetCursor((rbip->infoptr+mourb)->x1+1, (rbip->infoptr+mourb)->y);
        }
       return TF_ACCEPTED;                  /* Ereignis in jedem Fall akzeptieren */
      }
     else                                            /* unbekanntes Mausereignis */
      return TF_WEITER;
    }
  }
 else                  /* RB nicht aktiv oder Mausknopf außerhalb losgelassen */
  {
   if  ( ev & EV_LEFT_PRESS ) /* wurde der linke Mausbutton niedergedrückt? */
    if ( ( mourb = rbi_maus( rbip, x, y ) ) != -1  &&
         (rbip->infoptr+mourb)->enabled  )
     return TF_MAUS;              /* Maus über dem Button und Button enabled */
   return TF_WEITER;                                  /* Mausereignis abweisen */
  }
}

/***************************************************************************
*  Funktion         : r b _ c a n                                          *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um festzustellen, ob   *
*                     das Dialogfeld bereit ist, aktiviert zu werden.       *
*  Eingabe-Parameter: RBIP = der Zeiger, der beim Aufruf von rb_start zurück-*
*                            geliefert wurde.                               *
*  Return-Wert      : TRUE, wenn das Dialog-Feld aktiviert werden kann,     *
*                     sonst FALSE                                           *
***************************************************************************/

BOOL rb_can( RBIP rbip )
{
 return TRUE;
}

/***************************************************************************
*  Funktion         : r b _ a k t i v                                      *
**------------------------------------------------------------------------**
*  Aufgabe          : Wird vom Scheduler aufgerufen, um das Dialog-Feld von *
*                     seiner Aktivierung in Kenntnis zu setzen.             *
*  Eingabe-Parameter: RBIP = der Zeiger, der beim Aufruf von rb_start zurück-*
*                            geliefert wurde.                               *
*                     WHY  = warum wird das Feld aktiviert                  *
*  Return-Wert      : keiner                                                *
***************************************************************************/

void rb_aktiv( RBIP rbip, BYTE why )
{
 RBINFO * riptr;                 /* Zeiger auf Element im internen RB-Vektor */

 rbip->aktiv = TRUE;                                   /* neuen Status merken */
 rbip->mouout = FALSE; /* Mausknopf nicht außerhalb der Buttons losgelassen */
 justrel = FALSE;                           /* noch keine Umschaltung vorgenommen */
 riptr = rbip->infoptr + *rbip->varptr;   /* Zeiger auf akt. Element setzen */
 VioSetCursor( riptr->x1+1, riptr->y );  /* Cursor auf den aktuellen Button */
}
```

▶▶ *Listing 4: DLGDEMO.C*

```c
/**************************************************************************/
/*                           D L G D E M O . C                            */
/*------------------------------------------------------------------------*/
/*    Aufgabe        : Demonstriert die Erstellung und Verwaltung von Dia- */
/*                     logboxen durch die Module DLG, DLGEDIT und DLGAB,   */
/*                     DLGPB und DLGRB.                                    */
/*------------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                     */
/*    entwickelt am  : 13.07.1989                                          */
/*    letztes Update : 19.09.1989                                          */
/*------------------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] DLGDEMO.C DLG.C DLGAB.C DLGEDIT.C  */
/*                                      DLBPB.C DLGRB.C VIO.C KBM.C        */
/**************************************************************************/

/*-- Include-Dateien einbinden -------------------------------------------*/

#include "dlg.h"

/*-- Forward-Funktionsdeklarationen --------------------------------------*/

void paint( void );

DLGCOL demofarbe =                 /* Farben in der Dialogox für Color-Modus */
 {
  COL( WEISS, CYAN ),                              /* Rahmenfarbe            */
  COL( SCHWARZ, CYAN ),                            /* normale Zeichen        */
  COL( GELB, SCHWARZ ),                            /* hervorgehobene Zeichen */
  COL( ROT, CYAN ),                                /* Hotkeys                */
  COL( HGRAU, CYAN ),                              /* inaktive Felder        */
  COL( HGRAU, CYAN )                               /* inaktive Felder, hotkey */
 };

/*-- Daten für Eingabefeld #1 --------------------------------------------*/

BYTE f1buf[21] = "";                                       /* Eingabepuffer */
EMENGE f1menge;                                             /* Eingabemenge */

EDITFELD tb_dlgf1 =
 {
  TRUE, 3, 2, 20, 20, "alpha#numerisch : ", f1buf, f1menge
 };

/*-- Daten für Eingabefeld #2 --------------------------------------------*/

BYTE f2buf[81] = "";                                       /* Eingabepuffer */
EMENGE f2menge;                                             /* Eingabemenge */

EDITFELD tb_dlgf2 =
 {
  TRUE, 3, 4, 80, 23, "#Dateiname      : ", f2buf, f2menge
 };

/*-- Daten für Eingabefeld #3 --------------------------------------------*/

BYTE f3buf[11] = "1,23";                                   /* Eingabepuffer */
EMENGE f3menge;                                             /* Eingabemenge */

EDITFELD tb_dlgf3 =
 {
   TRUE, 3, 6, 10, 10, "#Integer        : ", f3buf, f3menge
 };

/*-- Push-Button Nummer 1 ------------------------------------------------*/

BOOL verify = TRUE;      /* Variable, die mit dem Push-Button verbunden ist */

ONEPB tb_pb1 = { TRUE, 3, 8, "#Verify", &verify };

/*-- Push-Button Nummer 2 ------------------------------------------------*/

BOOL loadinit = TRUE;    /* Variable, die mit dem Push-Button verbunden ist */

ONEPB tb_pb2 = { TRUE, 18, 8, "#Start mit AUTOEXEC.BAT", &loadinit };

/*-- Daten für die Radio-Buttons -----------------------------------------*/

BYTE textformat = 3;                               /* aktueller Radio-Button */

ONERB rbs[] = {                /* Beschreibung der verschiedenen Radio-Buttons */
               { TRUE , 49,  2, "#Wordstar"        },
               { TRUE , 49,  3, "#MS-Word"         },
               { TRUE , 49,  4, "AS#CII"           },
               { TRUE , 49,  5, "#Textomat Plus"   },
               { FALSE, 49,  6, "Word-#Perfect"    },
               { TRUE , 49,  7, "DD#E"             }
              };
```

```
RBGROUP tb_rb = { 6, &textformat, rbs };

/*— Daten für die Action-Buttons———————————————————— */

ONEAB tb_absdef[] =
 {
  { TRUE,  3, 10, "   OK   ",    NOKEY },
  { TRUE, 26, 10, "ESC = #Abbruch", ESC },
  { TRUE, 54, 10, "F1 = #Hilfe",     F1 },
 };

ABGROUP tb_abs = { 3, 0, tb_absdef };

/*— Beschreibung der einzelnen Dialogfelder ——————————————*/

DLGDATEN tb_dfvek[] =
 {
  EDIT(tb_dlgf1),                 /* die Reihenfolge bestimmt auch die  */
  EDIT(tb_dlgf2),                 /* Reihenfolge, mit der die einzelnen */
  EDIT(tb_dlgf3),                 /* Felder mit TAB und mit SHIFT TAB   */
  PUSHBUT(tb_pb1),                /* angesprungen werden können         */
  PUSHBUT(tb_pb2),
  RADIOBUT(tb_rb),
  ACTBUT(tb_abs)
 };

/*— Beschreibung der Dialogbox ——————————————————————*/

DIGDES testbox = { 6, 8, 69, 12, 7, paint, tb_dfvek };

/**********************************************************************
*  Funktion         : p a i n t                                       *
**____________________________________________________________________**
*  Aufgabe          : Wird von der Dialog-Prozedur DlgStart während des Auf- *
*                     baus der Demo-Dialogbox aufgerufen.             *
*  Eingabe-Parameter: keine                                           *
*  Return-Wert      : keiner                                          *
**********************************************************************/

void paint( void )
{
 VioPrint( VL(28), VO(0), F(hi), FALSE, " DEMOBOX " );    /* Titel ausgeben */
 VioPrintf( VL(0), VU(-2), F(dlgbox), FALSE, "├%s┤",
            VioStrep( '-', VR(-1) - VL(1) + 1 ) );
 DlgBox( 1, 1, 45, 7, EINRA, " EDIT-Felder ", FALSE );
 DlgBox( 47, 1, 21, 8, EINRA, " Radio Buttons ", FALSE );
}


/*===================================================================*/
/*== Hauptprogramm ==================================================*/
/*===================================================================*/

void main( void )
{
 BYTE    auswahl;          /* nimmt Nr. des ausgewählten Action-Buttons auf */
 BEREICH bereich;            /* Koordinaten des OK-Buttons im Hilfe-Fenster */
 BOOL    ende;

 VioInit();                         /* die SAA-Modul VIO und KBM initialisieren */
 KbmInit();

 SetMengeAN( f1menge );              /* die drei Eingabemengen initialisieren  */
 SetMengeDatei( f2menge );
 SetMengeInt( f3menge );

 if ( VioIsColor() )                     /* ist ein Color-Adapter angeschlossen? */
  dlgcol = demofarbe;       /* Ja, Farbeinstellungen für Dialogbox vornehmen */

 VioClearScreen( VioIsColor() ? COL( WEISS, BLAU ) : NORMAL );
 VioClear( 0, 0, anzcol-1, 0, VioIsColor() ? COL( WEISS, ROT ) : INVERS );
 VioPrint( 22, 0, VioIsColor() ? COL( WEISS, ROT ) : INVERS , FALSE,
          "DLGDEMO - (c) 1989 by MICHEAL TISCHER");
 MouShowMouse();                                    /* Maus-Cursor anzeigen */

 do                                                     /* Eingabeschleife */
  {
   auswahl = DlgStart( &testbox ); /* Dlgbox aufb., Eingaben entgegennehmen */
   if ( auswahl == 2 )                              /* Hilfetaste betätigt? */
    {                                                               /* Ja */

     /*— Hilfe-Fenster öffnen und aufbauen ————————————————————*/

     MouHideMouse();                               /* Maus-Cursor ausblenden */
     VioWinOpen( 40, 16, 73, 24 );
     VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, F(dlgbox) );
     VioClear( VL(1), VO(1), VR(-1), VU(-1), F(nm) );
     VioPrint( VL(13), VO(0), F(hi), FALSE, " Hilfe! " );
     VioPrint( VL(7), VO(2), F(nm), FALSE, "Bitte \x11─┘ betätigen" );
     VioFrame( VL(21), VO(5), VR(-3), VU(-1), EINRA, F(dlgbox) );
     VioPrint( VL(25), VO(6), F(nm), FALSE, "OK" );
     VioSetCursor( VL(25), VO(6) );
     MouPushPara();                                 /* Mausparameter sichern */
     bereich.x1 = VL(21);                   /* Position des OK-Felds setzen */
     bereich.y1 = VO( 5);
     bereich.x2 = VR(-3);
     bereich.y2 = VU(-1);
     bereich.ptr_mask = MouPtrMask( PTRDIFCHAR( 251 ), PTRDIFCOLB( 0x70 ) );
     MouSetBereich( 1, &bereich );                 /* das OK-Feld definieren */
     ende = FALSE;
     MouShowMouse();

     do
      {                                                 /* auf Event warten */
       if ( KbmEventWait( EV_LEFT_PRESS | EV_KEY_AVAIL ) & EV_KEY_AVAIL )
        ende = ( KbdGetKey() == CR );               /* Taste. Ist es Return? */
       else                                   /* linker Mausknopf niedergedrückt */
        if ( MouGetBereich() == 0 )                          /* im OK-Feld? */
         {                           /* Ja, auf Loslassen des Buttons warten */
          KbmEventWait( EV_LEFT_REL );
          ende = TRUE;
         }
      }
     while ( !ende );
     MouPopPara();                          /* alte Mausparameter restaurieren */
     MouHideMouse();                               /* Maus-Cursor ausblenden */
     VioWinClose( TRUE );
     MouShowMouse();
    }
  }
 while ( auswahl!=0  &&  auswahl!=1 );


 MouHideMouse();                            /* Maus-Cursor wieder ausblenden */
 VioSetCursor( 0, 0 );                    /* Cursor in obere linke Bildschirmecke */
}
```

Microsoft C-Kurs für Umsteiger, Teil 2:

Programmieren in C: Funktionen, Operatoren und einfache Pointer

In der letzten Folge haben wir die Grundstrukturen der Sprache C kennengelernt und verschiedene Programme geschrieben. Der jetzige Teil hat drei Schwerpunkte: Funktionen, Operatoren und einfache Pointer.

1. Einleitung

Alle Programme, die wir im letzten Teil erstellt haben, bestanden nur aus einer einzigen Funktion, der main-Funktion. In diesem Teil werden wir sehen, daß ein Programm aus einer beliebigen Anzahl von Funktionen bestehen kann.

Funktionen sollten so implementiert werden, daß Sie möglichst allgemein verwendbar sind. Die Funktion wird getrennt kompiliert und getestet. Wir werden uns mit der getrennten Kompilation und der Archivierung von Objektdateien in Bibliotheken beschäftigen (Library Manager).

Eine Funktion kann von dem Entwickler wie eine Black Box betrachtet werden. Sie wird mit bestimmten Werten versorgt und erledigt eine definierte Aufgabe. Die technischen Details der Implementierung bleiben dem Entwickler verborgen. Das Programm wird aus einer Zahl getrennt entwickelter Komponenten wie in Fertigbauweise zusammengesetzt.

Um den Informationsaustausch zwischen der aufrufenden und der aufgerufenen Funktion zu verstehen, werden wir uns mit den Grundlagen der Adreßbehandlung beschäftigen.

Einen weiteren Schwerpunkt bilden die Operatoren. Wir werden uns hauptsächlich mit Bitoperatoren beschäftigen und Probleme der Auswertungspriorität und -reihenfolge behandeln.

Am Ende des Seminars werden Sie in der Lage sein, ein C-Programm aus einer beliebigen Anzahl von Funktionen zu erstellen. Der Stoff wird anhand von drei Übungsaufgaben verfestigt, die wir zu Beginn der nächsten Folge besprechen.

1.1 Musterlösungen

Bevor wir uns den Funktionen und Operatoren zuwenden, zunächst die Musterlösungen und Kommentare zu den Aufgaben der letzten Folge. Es gibt zahlreiche Lösungsmöglichkeiten für ein gestelltes Problem. Wir verwenden immer nur Lösungswege, die mit den bereits besprochenen Sprachkonstrukten realisierbar sind. In vielen Fällen existieren elegantere oder schnellere Möglichkeiten. Wir werden auf diese Punkte hinweisen und entsprechende Modifikationen an den Programmen vornehmen, wenn wir die benötigten Sprachkonstrukte behandelt haben.

Es sollte ein Programm erstellt werden, daß einen Rahmen auf den Bildschirm zeichnet. Die Musterlösung (*Abbildung 1*) erstellt einen Rahmen mit den Abmessungen (VONZEILE/VONSPALTE) bis (BISZEILE/BISSPALTE). Das Programm bedient sich der Grafikzeichen des erweiterten IBM-Zeichensatzes. Beachten Sie bitte, daß die Zeichencodes hexadezimal eingegeben sind (führendes 0x). Es zeichnet zunächst die horizontalen und dann die vertikalen Linien. Anschließend werden die Eckstücke eingesetzt. Durch das Makro RAHMEN wird festgelegt, ob ein einfacher (1) oder doppelter (2) Rahmen gezeichnet werden soll.

```c
#include    <stdio.h>
#include    <conio.h>

#include    "ckurs.h"

#define RAHMEN      2
#define VONZEILE    5
#define VONSPALTE   5
#define BISZEILE   20
#define BISSPALTE  75

main ()
{
        int     i;

        ANSI_CLR();
        /* obere horizontale Linien ausgeben */

        ANSI_CURPOS(VONZEILE, VONSPALTE+1);

        for (i= VONSPALTE+1; i<BISSPALTE; i++)
                if (RAHMEN==EINFACH)
                        printf("%c", HORIZONTAL_EINFACH);
                else
                        printf("%c", HORIZONTAL_DOPPELT);

        /* untere horizontale Linien ausgeben */
        ANSI_CURPOS(BISZEILE,VONSPALTE+1);
        for (i= VONSPALTE+1; i<BISSPALTE; i++)
                if (RAHMEN==EINFACH)
                        printf("%c", HORIZONTAL_EINFACH);
                else
                        printf("%c", HORIZONTAL_DOPPELT);

        /* linker und rechte vertikale Linie ausgeben */
        for (i= VONZEILE+1; i<BISZEILE; i++)
        {
        /* oben */
                ANSI_CURPOS(i,VONSPALTE);

                if (RAHMEN==EINFACH)
                        printf("%c", VERTIKAL_EINFACH);
                else
                        printf("%c", VERTIKAL_DOPPELT);
        /* unten */
                ANSI_CURPOS(i,BISSPALTE);

                if (RAHMEN==EINFACH)
                        printf("%c", VERTIKAL_EINFACH);
                else
                        printf("%c", VERTIKAL_DOPPELT);
        }

        /* linke, obere Ecke ausgeben */
        ANSI_CURPOS(VONZEILE,VONSPALTE);
        if (RAHMEN==EINFACH)
                printf("%c", LO_ECKE_EINFACH);
        else
                printf("%c", LO_ECKE_DOPPELT);

        /* rechte, obere Ecke ausgeben */
        ANSI_CURPOS(VONZEILE,BISSPALTE);

        if (RAHMEN==EINFACH)
                printf("%c", RO_ECKE_EINFACH);
        else
                printf("%c", RO_ECKE_DOPPELT);

        /* linke, untere Ecke ausgeben */
        ANSI_CURPOS(BISZEILE,VONSPALTE);

        if (RAHMEN==EINFACH)
                printf("%c", LU_ECKE_EINFACH);
        else
                printf("%c", LU_ECKE_DOPPELT);

        /* rechte, untere Ecke ausgeben */
        ANSI_CURPOS(BISZEILE,BISSPALTE);

        if (RAHMEN==EINFACH)
                printf("%c", RU_ECKE_EINFACH);
        else
                printf("%c", RU_ECKE_DOPPELT);

        getch ();
}
```

◄◄ *Abbildung 1: Musterlösung von Aufgabe 1.*

Die zweite Aufgabe bestand darin, eine Multiple-Choice-Auswahl am Bildschirm anzuzeigen und den Benutzer eine beliebige Anzahl von Optionen ein- und ausschalten zu lassen. Die Ausgabe der Texte ist hier sehr einfach über Cursorpositionierung (ANSI_CURPOS-Makro) und anschließende Textausgabe implementiert. Wir werden im Rahmen dieser Folge noch elegantere Möglichkeiten kennenlernen.

Interessant ist die Verwaltung der verschiedenen Zustände. Von den vier Optionen kann jede ein- oder ausgeschaltet sein. Die Zustände werden am besten in einem Array verwaltet. Wir wählen ein Array aus INT-Werten (int wahl [ANZAHL]), das auf 0 (für nicht gewählt) initialisiert wird. Anschließend fragt das Programm in einer Schleife solange die Tastatur ab, bis [Return] gedrückt wird. Die Leertaste schaltet die Optionen an- oder aus. Zum Wechsel zwischen den Feldern benutzen wir die [Tab]-Taste.

```c
/* Programm mit check-box, Musterloesung
*/

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <conio.h>
#include    "ckurs.h"

#define CR      13      /* Code fuer Carriage Return */
#define BLANK   32      /* Code fuer Blank           */
#define TAB     9       /* Code fuer Tab-Taste       */
#define ZEILE   10
#define SPALTE  20
#define ANZAHL  5

main ()
{
        int     taste;          /* eingegebene Taste      */
        int     aktiv;          /* aktuelles Feld         */
        int     wahl[ANZAHL];   /* Feld fuer Markierungen */
        int     i;              /* Laufvariable           */

/* Bildschirm loeschen
*/
        ANSI_CLR();

/* Attribut setzen
*/
        ANSI_ATTR(ANSI_NORMAL);

/* Texte ausgeben
*/
        ANSI_CURPOS(ZEILE-2, SPALTE);
        printf ("%s", "In der Programmiersprache C");

        ANSI_CURPOS(ZEILE, SPALTE);
        printf ("%s", "[ ]  endet jede Anweisung mit einem Semikolon");

        ANSI_CURPOS(ZEILE+1, SPALTE);
        printf ("%s", "[ ]  gibt es Schlüsselwörter für Ein- und Ausgabe");

        ANSI_CURPOS(ZEILE+2, SPALTE);
        printf ("%s", "[ ]  beginen Felder bei 0");

        ANSI_CURPOS(ZEILE+3, SPALTE);
        printf ("%s", "[ ]  ist der Vergleichsoperator für Identität \"=\"");

        ANSI_CURPOS(ZEILE+4, SPALTE);
        printf ("%s", "[ ]  sind die Anweisungen a=a+3 und a+=3 identisch");

        ANSI_CURPOS(ZEILE+6, SPALTE);
        printf ("%s", "Leertaste = Auswahl");

        ANSI_CURPOS(ZEILE+7, SPALTE);
        printf ("%s", "TAB       = nächstes Feld");

        ANSI_CURPOS(ZEILE+8, SPALTE);
        printf ("%s", "Eingabe   = alle Fragen beantwortet");

/* Cursor auf erstes Feld positionieren
*/
        ANSI_CURPOS (ZEILE, SPALTE+1);

/* Das Feld wahl initialisieren
*/
        for (i = 0; i < ANZAHL; i++)
                wahl[i] = 0;

/* Eingabe erwarten
*/
        for (taste= 0, aktiv= 0; taste!=CR; )
        {
                taste= getch ();

                switch (taste)
                {
/* Leertaste gedrueckt ? ==> Lichtbalken wechseln
*/
                case BLANK :
/* Cursor positionieren
*/
                        ANSI_CURPOS(ZEILE+aktiv, SPALTE+1);

                        if (wahl[aktiv])
                        {
                                printf ("%s", " ");
                                wahl[aktiv] = 0;
                        }
                        else
                        {
                                printf ("%s", "X");
                                wahl[aktiv] = 1;
                        }
/* Nach der Ausgabe, Cursor erneut positionieren
*/
                        ANSI_CURPOS (ZEILE+aktiv, SPALTE+1);

                break;

/* Tab gedrueckt ? ==> Naechstes Feld
*/
                case TAB :
                        if (aktiv == ANZAHL-1)
                                aktiv = 0;
                        else
                                aktiv++;

/* Cursor auf naechstes Feld positionieren
*/
                        ANSI_CURPOS (ZEILE+aktiv, SPALTE+1);

                break;
                }
        }

/* Bildschirm loeschen
*/
        ANSI_CLR();

/* Ausgabe, ob die angekreuzten Optionen richtig sind
*/
        if (wahl[0]==1 && wahl[1]==0 && wahl[2]==1 && wahl[3]==0 && wahl[4]==1)
                printf ("Bravo. Sie haben die Fragen richtig beantwortet\n", i+1);
        else
        {
                printf ("Sie haben die Fragen leider nicht richtig beantwortet\n");
                printf ("Sie hätten die Fragen 1, 3 und 5 ankreuzen müssen\n");
        }
}
```

◄ *Abbildung 2: Musterlösung checkbox.*

▶ *Abbildung 3: Musterlösung Radiobutton.*

▶▶ *Abbildung 4: ckurs.h.*

```
/* Programm mit check-box, Musterloesung Aufgabe 3, Abbildung3
*/

#include    <stdio.h>
#include    <conio.h>
#include    "ckurs.h"

#define CR      13      /* Code fuer Carriage Return */
#define BLANK   32      /* Code fuer Blank           */
#define TAB      9      /* Code fuer Tab-Taste       */
#define ZEILE   10
#define SPALTE  20
#define ANZAHL  5

main ()
{
        int     taste;          /* eingegebene Taste      */
        int     aktiv;          /* aktuelles Feld         */
        int     wahl[ANZAHL];   /* Feld fuer Markierungen */
        int     i;                  /* Laufvariable               */

        /* Bildschirm loeschen */
        ANSI_CLR();
        /* Attribut setzen */
        ANSI_ATTR(ANSI_NORMAL);
        /* Texte ausgeben */
        ANSI_CURPOS(ZEILE-2, SPALTE);
        printf ("%s", "In der Programmiersprache C");

        ANSI_CURPOS(ZEILE, SPALTE);
        printf ("%s", "( )  endet jede Anweisung mit einem Komma");

        ANSI_CURPOS(ZEILE+1, SPALTE);
        printf ("%s", "( )  gibt es Schlüsselwörter für Ein- und Ausgabe");

        ANSI_CURPOS(ZEILE+2, SPALTE);
        printf ("%s", "( )  beginen Felder bei 0");

        ANSI_CURPOS(ZEILE+3, SPALTE);
        printf ("%s", "( )  ist der Vergleichsoperator für Identität \"=\"");

        ANSI_CURPOS(ZEILE+4, SPALTE);
        printf ("%s", "( )  sind die Anweisungen a=a+2 und a++ identisch");

        ANSI_CURPOS(ZEILE+6, SPALTE);
        printf ("%s", "Leertaste = Auswahl");

        ANSI_CURPOS(ZEILE+7, SPALTE);
        printf ("%s", "TAB       = nächstes Feld");

        ANSI_CURPOS(ZEILE+8, SPALTE);
        printf ("%s", "Eingabe   = alle Fragen beantwortet");

        /* Cursor auf erstes Feld positionieren */
        ANSI_CURPOS (ZEILE, SPALTE+1);

        /* Das Feld wahl initialisieren */
        for (i = 0; i < ANZAHL; i++)
                wahl[i] = 0;

        /* Eingabe erwarten */
        for (taste= 0, aktiv= 0; taste!=CR; )
        {
                taste= getch ();
                switch (taste)
                {
                /* Leertaste gedrueckt ? ==> Lichtbalken wechseln */
                case BLANK :

                        /* die ausgewaehlte Frage zuruecksetzen */
                        for (i= 0; i<ANZAHL; i++)
                        {
                                ANSI_CURPOS(ZEILE+i,SPALTE+1);
                                printf ("%s", " ");
                                wahl[aktiv]= 0;
                        }

                        if (wahl[aktiv])
                        {
                                printf ("%s", " ");
                                wahl[aktiv] = 0;
                        }

                        /* Cursor positionieren und Selektion ausgeben */
                        ANSI_CURPOS(ZEILE+aktiv, SPALTE+1);
                        printf ("%c", '\371');
                        wahl[aktiv] = 1;

                        /* Nach der Ausgabe, Cursor erneut positionieren */
                        ANSI_CURPOS (ZEILE+aktiv, SPALTE+1);
                break;

                /* Tab gedrueckt ? ==> Naechstes Feld */
                case TAB :
                        if (aktiv == ANZAHL-1)
                                aktiv = 0;
                        else
                                aktiv++;

                        /* Cursor auf naechstes Feld positionieren */
                        ANSI_CURPOS (ZEILE+aktiv, SPALTE+1);

                break;
                }
        }

        /* Bildschirm loeschen */
        ANSI_CLR();
        /* Ausgabe, ob die angekreuzten Optionen richtig sind */
        if (wahl[2]==1)
                printf ("Bravo. Sie haben die Fragen richtig beantwortet\n", i+1);
        else
        {
                printf ("Sie haben die Fragen leider nicht richtig beantwortet\n");
                printf ("Sie hätten nur die Frage 3 ankreuzen dürfen\n");
        }
}
```

Die dritte Aufgabe ist eine Variation der zweiten. Es darf nur eine von verschiedenen Alternativen gewählt werden. Diese Auswahlart wird als Radio-Button bezeichnet. Ähnlich wie bei einem Radio kann nur ein Sendeknopf gedrückt sein.

```
/* Headerfile fuer C-Kurs */

#define ANSI_INVERS                     7
#define ANSI_NORMAL                     0
#define ANSI_CLR()                      printf ("\033[2J")
#define ANSI_CURPOS(zeile, spalte)      printf ("\033[%u;%uH",zeile, spalte)
#define ANSI_ATTR(attribut)             printf ("\033[%um", attribut)
#define ANSI_CUR_LEFT                   printf ("\033[D")

/* Tastencodes */
#define TAB     9
#define S_TAB   (15+256)
#define CR      13
#define ESC     27
#define BLANK   32

/* einfache Blockgrafik */
#define HORIZONTAL_EINFACH      0xc4
#define VERTIKAL_EINFACH        0xb3
#define LO_ECKE_EINFACH         0xda
#define RO_ECKE_EINFACH         0xbf
#define LU_ECKE_EINFACH         0xc0
#define RU_ECKE_EINFACH         0xd9

/* doppelte Blockgrafik */
#define HORIZONTAL_DOPPELT      0xcd
#define VERTIKAL_DOPPELT        0xba
#define LO_ECKE_DOPPELT         0xc9
#define RO_ECKE_DOPPELT         0xbb
#define LU_ECKE_DOPPELT         0xc8
#define RU_ECKE_DOPPELT         0xbc

/* Rahmenarten */
#define EINFACH                 1
#define DOPPELT                 2
```

Das Beispielprogramm ist fast genauso aufgebaut wie Beispiel 2. Allerdings muß die zuvor gewählte Option vor Auswahl einer neuen wieder gelöscht werden. Das Beispiel geht einen einfachen Weg. Es setzt alle Optionen auf 0 bevor eine neue angekreuzt wird.

Entwickeln Sie eine elegantere Möglichkeit, indem sie sich den selektierten Punkt merken und nur diesen gezielt zurücksetzen.

Das Programm vier ist eine Zusammenfassung aus Aufgabe 1 und 3. Eine Kommentierung erübrigt sich. Die *Abbildung 4* enthält den aktuellen Stand des Headerfiles ckurs.h, wie es für die Musterlösungen benötigt wird.

2. Funktionen

Dem aufmerksamen Leser wird nicht entgangen sein, daß die Beispielprogramme redundanten Code enthalten. In Aufgabe 1 und 4 wird der Code für das Zeichnen eines Rahmens zweimal benötigt. Dies ist ein sehr ineffizientes Verfahren. Man verschwendet Speicherplatz und die Lösung ist fehleranfällig. Außerdem ist redundanter Code sehr wartungsfeindlich. Sie haben sicherlich schon bemerkt, daß die Funktion zum Zeichnen der Linien recht langsam ist. Wenn man sich nun ein schnelleres Verfahren ausdenken würde, müßte der Code in Ausgabe 1 und Aufgabe 2 geändert werden. Änderungsaufwand und Fehleranfälligkeit steigen mit der Coderedundanz.

Es bietet sich an, den Code zum Zeichnen eines Rahmens in eine Funktion auszulagern, die dann bei Bedarf nur noch aufgerufen wird. Diese Funktion läßt sich getrennt testen und warten. Verbesserungen, was die Performance angeht, können durchgeführt werden, ohne daß die aufrufenden Funktionen hiervon berührt wären. Die Details der Implementierung sind der aufrufenden Funktion verborgen. Sie betrachtet drawbox (so soll die Funktion heißen) als Black Box.

Der Informationsaustausch zwischen der aufrufenden Funktion (z.B. main) und drawbox() erfolgt über die Schnittstelle der Funktion drawbox(). Solange die Schnittstelle zwischen zwei Funktionen unverändert bleibt und die Funktionen diese Schnittstelle nicht umgehen (z.B. über globale Variablen), sind sie voneinander vollkommen unabhängig.

2.1 Wertübergabe

Die Schnittstelle der Funktion drawbox hat folgendes Aussehen:

```
int drawbox (vonzeile, vonspalte, biszeile, bisspalte, box)
int vonzeile;
int vonspalte;
int biszeile;
int bisspalte;
int box;
{
}
```

Die Funktion wird mit den Koordinaten der linken oberen und der rechten unteren Ecke versorgt. Über den Parameter box wird mitgeteilt, ob ein einfacher (EINFACH) oder ein doppelter (DOPPELT) Rahmen gezeichnet werden soll. Die Funktion wird dann von main() so aufgerufen:

```
main ()
{
        if (! drawbox (1,1,24,80,EINFACH))
                printf ("Falsche Parameter !\n");

}
```

An drawbox werden fünf Werte übergeben, die innerhalb der Funktion über die deklarierten Parameter angesprochen werden können. Wichtig ist hierbei, das C standardmäßig eine sogenannte Wertübergabe (= call by value) vornimmt, daß heißt drawbox() arbeitet auf einer Wertkopie und nicht auf der Originalvariablen. Ein Beispiel für die Funktion zeigt *Abbildung 5*.

2.2 Returnwert

Drawbox überprüft die übergebenen Parameter auf Gültigkeit und liefert eine 0, wenn ungültige Werte übergeben wurden. Der erfolgreiche Abschluß wird durch eine 1 angezeigt. Der Returnwert von drawbox() kann direkt innerhalb der aufrufenden Funktion abgefragt werden. Wenn die Funktion 0 liefert, wird dieser Wert durch den logischen Negationsoperator ! zum Wert 1. Jeder Wert ungleich 0 gilt in C als wahr. Das Programm verzweigt in die If-Anweisung.

Das Return-Statement kann an beliebiger Stelle innerhalb einer Funktion stehen. Es beendet die Funktion und liefert den entsprechenden Wert an die aufrufende Funktion. Es können nur elementare Datentypen, also keine Arrays oder Strukturen zurückgeliefert werden.

2.3 Getrennte Übersetzung

Die Funktion drawbox ist nun ein allgemein verwendbares Werkzeug, das ordentlich in unserer Werkzeugkiste verstaut werden muß. Zunächst übersetzen wir die Funktion drawbox() mit

```
/* Funktion drawbox() mit einfacher Fehlerbehandlung
*/

#include    <stdio.h>
#include    "ckurs.h"

int drawbox (vonzeile, vonspalte, biszeile, bisspalte, box)
int vonzeile;
int vonspalte;
int biszeile;
int bisspalte;
int box;
{
        int     i;

        if (vonzeile  >= biszeile ||
            vonspalte >= bisspalte ||
            biszeile  >  25 ||
            bisspalte >  80)
            return 0;

        /* obere horizontale Linien ausgeben */

        ANSI_CURPOS(vonzeile, vonspalte+1);
        for (i= vonspalte+1; i<bisspalte; i++)
                if (box==EINFACH)
                        printf("%c", HORIZONTAL_EINFACH);
                else
                        printf("%c", HORIZONTAL_DOPPELT);

        /* untere horizontale Linien ausgeben */
        ANSI_CURPOS(biszeile,vonspalte+1);

        for (i= vonspalte+1; i<bisspalte; i++)
                if (box==EINFACH)
                        printf("%c", HORIZONTAL_EINFACH);
                else
                        printf("%c", HORIZONTAL_DOPPELT);


        /* obere und untere vertikalen Linien ausgeben */
        for (i= vonzeile+1; i<biszeile; i++)
        {
                /* oben */
                ANSI_CURPOS(i,vonspalte);

                if (box==EINFACH)
                        printf("%c", VERTIKAL_EINFACH);
                else
                        printf("%c", VERTIKAL_DOPPELT);

                /* unten */
                ANSI_CURPOS(i,bisspalte);

                if (box==EINFACH)
                        printf("%c", VERTIKAL_EINFACH);
                else
                        printf("%c", VERTIKAL_DOPPELT);
        }

        /* linke, obere Ecke ausgeben */
        ANSI_CURPOS(vonzeile,vonspalte);

        if (box==EINFACH)
                printf("%c", LO_ECKE_EINFACH);
        else
                printf("%c", LO_ECKE_DOPPELT);

        /* rechte, obere Ecke ausgeben */
        ANSI_CURPOS(vonzeile,bisspalte);

        if (box==EINFACH)
                printf("%c", RO_ECKE_EINFACH);
        else
                printf("%c", RO_ECKE_DOPPELT);

        /* linke, untere Ecke ausgeben */
        ANSI_CURPOS(biszeile,vonspalte);

        if (box==EINFACH)
                printf("%c", LU_ECKE_EINFACH);
        else
                printf("%c", LU_ECKE_DOPPELT);

        /* rechte, untere Ecke ausgeben */
        ANSI_CURPOS(biszeile,bisspalte);

        if (box==EINFACH)
                printf("%c", RU_ECKE_EINFACH);
        else
                printf("%c", RU_ECKE_DOPPELT);

        return 1;
}
```

◄ *Abbildung 5: Die Funktion drawbox.*

```
cl /c /W3 drawbox.c
```

Die Option W3 selektiert die höchste Warnungsstufe. Fehlende Prototypen werden hier mit einer Warnung angemahnt. Die Option /c unterbindet den anschließenden Linkvorgang.

Ergebnis des Compilerlaufs ist ein drawbox.obj, das den kompilierten Code enthält. Die Funktion kann nun in eine Bibliothek gestellt werden. Der Library-Manager (lib) wird wie folgt aufgerufen:

```
lib ckurs.lib +drawbox;
```

Das Objektmodul drawbox.obj wird in die Bibliothek ckurs.lib gestellt (+). Wenn die Bibliothek nicht vorhanden ist, wird sie angelegt. Eine

► *Abbildung 6: Aufruf von drawbox; getrennte Kompilierung.*

```
/* Falsche Parameterübergabe bei fehlenden Prototypen */

#include    <stdio.h>
#include    "ckurs.h"

/* Fehlender Prototyp fuehrt zu falscher Parameteruebergabe
   int drawbox (int vonzeile, int vonspalte, int biszeile, int bisspalte, int
rahmen);
*/

void main (void)
{
        long    zel;    /* Falscher Datentyp                    */

        zel = 10;

        if (! drawbox (zel, 20, 20, 75, DOPPELT))
                printf ("Parameterfehler bei drawbox !\n");
}
```

systematische Behandlung des Library-Managers finden Sie unter Abschnitt 6 (Utilities). Das Obj-File kann dann beispielsweise mit dem Objektfile abb6.obj über folgendes Linker-Kommando zu dem Programm abb6.exe gelinkt werden:

```
link abb6,,,ckurs.lib;
```

Eine detailliertere Behandlung des Linkers entnehmen Sie bitte dem Abschnitt 6 (Utilities).

2.4 Prototypen

Die Funktion drawbox() hat selbst überprüft, ob sie überhaupt mit gültigen Werten versorgt wurde.

Eine weitere Fehlermöglichkeit besteht darin, daß die Datentypen der formalen und aktuellen Parameter nicht zueinander passen. Derartige Fehler sind sehr schwer auffindbar. In *Abbildung 6* wird die Funktion drawbox() mit einem Long anstelle eines Int aufgerufen. Die aufrufende Funktion legt vier Bytes auf den Stack, die aufgerufene holt aber nur zwei Bytes ab. Die nächsten zwei Bytes werden als Spaltenwert interpretiert. Drawbox() kann keinen Fehler liefern, da die vom Stack abgeholten Werte durchaus sinnvoll sind. Dennoch wird natürlich ein falscher Rahmen gezeichnet. Anstelle eines Rahmens von (10,20) bis (20,75) wird ein Rahmen von (10,0) bis (20,20) gezeichnet. Der Stack ist wie folgt mit Parametern belegt:

Wert	Bedeutung	Interpretatiton durch Drawbox()
10	long, Byte 1 und 2	vonzeile
0	long, Byte 3 und 4	vonspalte
20	Startspalte	biszeile
20	Endzeile	bisspalte
75	Endspalte	Rahmenart
2	Rahmenart	wird nicht verwertet.

Um derartige Fehler aufzudecken, werden sogenannte Prototypen eingeführt. Ein Prototyp ist die formale Definition einer Schnittstelle einer Funktion, die dem Compiler bekannt gemacht wird. Der Compiler kann dann bei einem Aufruf der Funktion die aktuellen Datentypen der Parameter mit den formalen vergleichen. Wird eine Unstimmigkeit festgestellt, wird eine entsprechende Fehlermeldung erzeugt.

Der Prototyp von drawbox() sieht folgendermaßen aus:

```
extern int drawbox (int vonzeile,int vonspalte,int biszeile,
int bisspalte,int box);
```

Wenn nun die Funktion mit einem falschen Datentyp aufgerufen wird, bemerkt der Compiler diesen Fehler und meldet:

```
abb6.c (15): Warning C4051: Data conversion
```

Im folgenden werden die Unterschiede zwischen dem Funktionsprototypen, der Funktionsdefinition und dem Aufruf der Funktion dargestellt.

```
/* Funktionsprototyp = formale Schnittstelle */

extern int drawbox (int vonzeile, int vonspalte, int biszeile,
int bisspalte, int box);
```

Bisher wurden Schnittstellen definiert, in dem nach dem Funktionsnamen in Klammern durch Komma getrennt die Variablen aufgeführt wurden. Anschließend wurden die Datentypen der Variablen angegeben, gefolgt vom Funktionsrumpf (siehe oben). In der ANSI-Norm wurde definiert, daß die Datentypen auch innerhalb der Klammern angegeben werden können:

```
/* Funktionsdefinition */

int drawbox (int vonzeile, int vonspalte, int biszeile, int bisspalte,
int rahmen)
{
}

/* Funktionsaufruf mit aktueller Parameterliste */

if (! drawbox (1,1,24,80,EINFACH))
   printf ("Falsche Parameter !\n");
```

In den folgenden Beispielen werden wir bei der Definition der Schnittstelle die ANSI-Konvention verwenden.

Zur Erinnerung: Die Funktionsprototypen können mit Hilfe des MS-C-Compilers automatisch erzeugt werden. Der Aufruf

```
cl /Zg abb5.c
```

liefert den Funktionprototypen

```
extern int drawbox (int vonzeile, int vonspalte, int biszeile,
int bisspalte, int rahmen);
```

3.0 C-Sprachelemente

3.1 Pointer

Ein wichtiges und erfahrungsgemäß schwieriges Kapitel in der Programmierung mit C sind Pointer. Wir werden uns in dieser und der nächsten Folge detailliert damit beschäftigen.

Pointer sind eigentlich ganz gewöhnliche Variablen, nur sind die Werte, die Sie beinhalten, Speicheradressen. Zur Verarbeitung von Pointern stellt C zwei Operatoren zur Verfügung: & und *.

Der Operator & ermittelt die Speicheradresse einer Variablen. Der Ausdruck

```
&wert
```

liefert also einen Pointer auf die Variable wert.

Der Operator * bewirkt das Gegenteil: Er liefert den Inhalt des Speicherbereichs, auf den ein Pointer zeigt.

Eine Pointervariable wird mit dem '*'- Operator deklariert:

```
int *pwert;
```

Die Variable pwert kann die Adresse (*) eines Integers speichern. Die Deklaration kann gelesen werden als:

Die Variable, auf die <pwert> zeigt; *pwert ist vom Typ int

oder

Die Variable <pwert> ist vom Typ Zeiger auf int (int *).

Die Variable pwert enthält nach der Deklaration zunächst einen undefinierten Wert und weist von daher (zufallsbedingt) auf irgendeinen Speicherbereich. Bevor mit pwert gearbeitet werden kann, muß die Variable initialisiert werden:

```
int wert;
int *pwert;

wert=5;
pwert=&wert;
*pwert+=2;
```

Es werden zwei Variablen deklariert. Wert ist ein Integer, pwert kann die Adresse auf einen Integer speichern, ist also vom Typ »Zeiger auf Integer«. Beide Variablen werden auf dem Stack angelegt. Nachdem wert auf 5 gesetzt wurde, weisen wir der Variablen pwert die Adresse von wert zu (&wert). Da wert vom Typ int ist und &wert vom Typ Adresse eines int, sind die Datentypen auf der linken und rechten Seite des Gleichheitszeichens identisch.

In pwert steht jetzt die Adresse von wert. Über den Operator '*' kann nun diese Adresse dereferenziert werden. Der Ausdruck pwert liefert die Adresse von wert, der Ausdruck *pwert liefert das, was an dieser Adresse steht, in diesem Fall der Wert 5.

Die Zuweisung

```
*pwert+=2;
```

ist hier gleichbedeutend mit

```
wert+=2;
```

An dieser Stelle möchten wir Sie auf einen ebenso beliebten wie verheerenden Fehler aufmerksam machen: Den Zugriff auf einen uninitialisierten Pointer.

Durch die Deklaration von pwert wird eine Variable vom Typ »Zeiger auf int« (int *) reserviert. Die Variable ist aber nicht initialisiert. Wenn man nun eine Operation wie

```
*pwert=5
```

ausführt, wird der Inhalt von pwert als gültige Adresse interpretiert und an diese Adresse wird die Zahl fünf geschrieben. In der Regel gehört dieser Speicherplatz, auf den pwert zeigt, nicht zum Programm. Unter Umständen stehen hier wichtige Informationen (z.B. Betriebssystem-Interrupts), die durch die Zuweisung zerstört oder verändert werden. Die Folge ist dann meistens ein Programmabsturz.

Bevor man also auf *pwert zugreifen kann, muß pwert auf einen zum Programm gehörenden Speicherplatz zeigen (z.B. pwert=&wert).

Der Fehler ist besonders heimtückisch, da er lange untentdeckt bleiben kann und scheinbar zufällig auftritt.

Bevor wir uns weiter mit den Zeigern befassen, ein Paar Worte über deren Nutzen. Zeiger haben im wesentlichen die folgenden Vorteile:

1. Performance, z.B. Effizienz beim Sortieren.
Sortierprobleme lassen sich oftmals sehr effizient mit Pointern realisieren. Angenommen, Sie müssen eine Namensliste alphabetisch sortieren. Die Namen sind alle unterschiedlich lang. Es ist empfehlenswert, hier ein Array aus Pointern zu deklarieren (siehe unten). Jeder Pointer zeigt dann auf einen Namen. Um nun eine Sortierung zu erstellen müssen nicht die Namen selbst bewegt, sondern nur die Pointer umgesetzt werden. Es ergibt sich eine erhebliche Performancesteigerung gegenüber Verfahren ohne Pointerstruktur.

2. Call by reference
Es ist oftmals effizienter, an eine Funktion nicht einen Wert, sondern eine Adresse zu übergeben. Eine Adreßübergabe ist bei großen Datenobjekten schneller. Innerhalb der aufgerufenen Funktion wird dann über einen Pointer auf die übergebene Adresse zugegriffen.

3. Systemsoftware
Bei dynamischen Datenobjekten wird Speicherplatz während der Laufzeit eines Programms angefordert. Das Programm erhält einen Pointer auf den Anfang des Speichers und kann so Informationen dort ablegen.

3.2 Die Typbindung von Pointern, Rechnen mit Pointern

Pointer haben eine Typbindung, d.h. sie zeigen auf einen bestimmten Datentyp. Der Grund dieser Typbindung ist zunächst nicht ganz einsichtig. Eine Adresse ist schließlich eine Adresse, ob dort nun ein int, ein char oder ein long steht. Die Bedeutung der Typbindung ist aber wichtig, sobald man mit Zeigern rechnet und den Inhalt abfragen möchte. Das Problem der Zeigerarithmetik wird anhand der *Abbildung 7* deutlich gemacht.

► *Abbildung 7: char und int Pointer.*

►► *Abbildung 8: main() mit Parametern.*

```
#include <stdio.h>
#include <string.h>

void main (void);


void main()
{
        char    name[20];
        char    *pchar;
        int             *pint;

        memset (name, '\0', sizeof(name));
        strcpy (name, "Lottermann");

        printf ("\n");
        for (pchar=name; *pchar != '\0'; pchar++)
                printf ("%c", (*pchar));

        printf ("\n");
        for (pint=name; *pint != '\0'; pint++)
                printf ("%c", *pint);

}
```

Das Programm deklariert ein Array aus 20 Char. Dort wird mit der Funktion strcpy() die Zeichenkette »Lottermann« abgelegt. Strings sind Arrays aus einzelnen char, die mit einem Nullbyte ('\0') enden (siehe auch Folge 1, Stringkonstanten). Arrays können nicht einander zugewiesen werden, deshalb wird hier die Funktion strcpy (string copy) genutzt. Sie gehört zu einer Reihe von Stringfunktionen, die im Kapitel »Library Guide« besprochen werden.

In zwei Schleifen werden nun der Characterpointer pchar und der Integerpointer pint auf den Anfang von name gesetzt. Anschließend wird zeichenweise über den Namen gelaufen und jedes Zeichen ausgegeben. Bei der ersten Schleife wird wie erwartet

```
Lottermann
```

ausgegeben. In der zweiten Schleife erscheint ein verstümmeltes

```
Ltemn
```

Was ist geschehen?

Bei der Erhöhung des Pointers pint (pint++) richtet sich der Compiler nach dem Basistyp, d.h. nach dem Datentyp, auf den pint zeigt. Dieser ist vom Typ Integer. Ein Integer ist unter MS-DOS 2 Byte groß. Die Inkrementierung um eine Basiseinheit (pint++) bewirkt also, daß die Adresse, auf die pint zeigt, um zwei Bytes erhöht wird. Es wird nur jeder zweite Buchstabe ausgegeben!

Noch ein Wort zu Arrays und Pointern. Der Name eines Arrays ist in C immer gleichbedeutend mit seiner Anfangsadresse. In dem Initialisierungsteil der For-Schleife können deshalb die Pointer pchar und pint auf den Anfang des Arrays gerichtet werden. Bei der Zuweisung pint = pchar weist uns aber der Compiler auf die Typinkompatibilität hin:

```
warning  C4049: indirection to different types
```

Es handelt sich nur um eine Warnung, das Programm wird trotzdem übersetzt. Um dieses Problem zu beheben, muß dem Compiler mitgeteilt werden, daß die Zuweisung gewollt ist. Dies wird in C mit Hilfe einer Cast-Operation realisiert:

```
#include <stdio.h>

void main (int argc, char *argv[], char *envp[]);

void main (int argc, char *argv[], char *envp[])
{
        int     i;

        printf ("\n*********** Kommandozeilenparameter *************\n");
        for (i=0; i<argc; i++)
                printf ("Argument %d : %s\n", i, argv[i]);

                printf ("\n************ Environmentvariablen ***************\n");
        for (i=0; envp[i] != NULL; i++)
                printf ("ENV-Var. %d : %s\n", i, envp[i]);
}
```

```
pint= (int *) pchar;
```

Umgangssprachlich formuliert würde man sagen: Sieh pchar als Pointer auf int an und weise die Adresse, auf die pchar zeigt, an pint zu, obwohl pchar ein Zeiger auf char ist.

In dem obigen Beispiel haben wir übrigens noch Glück im Unglück. Der Integerpointer unterschlägt zwar jeden zweiten Buchstaben, er findet aber wenigstens ein Ende, wenn er auf das Nullbyte stößt. Wir haben zuvor den ganzen Array mit Nullbytes gefüllt (memset), so daß uns pint auch mit Zweiersprüngen nicht davonlaufen kann.

3.3 Arrays aus Pointern

Pointer sind elementare Datentypen. Man kann folglich auch aus Pointern Arrays bilden. Das sieht dann beispielsweise so aus:

```
char *ptexte[4];
```

Ptexte ist ein Array aus 4 Pointern auf Char-Objekte. Eine solche Datenstruktur erlaubt die effiziente Verwaltung von Zeichenketten unterschiedlicher Länge gestattet.

Dies wird in C beim Programmstart ausgenutzt, in dem die Kommandozeilenparameter auf einem Pointer an die Funktion main() übergeben werden.

Bisher hatte unsere main()-Funktion immer keine Parameter:

```
main()
```

C kann aber auf portablem Wege Argumente aus der Kommandozeile verarbeiten. In *Abbildung 8* sehen Sie ein C-Programm, daß drei Parameter hat:

1. argc

Argc enthält die Anzahl der aktuell übergebenen Kommandozeilenparameter. C kann eine variable Anzahl von Parametern unterschiedlicher Länge verarbeiten. Argc ist immer größer gleich eins, da der Name des Programms selbst mit übergeben wird.

2. argv

Argv ist ein Array aus Pointern auf char. Das Array hat argc Elemente. Jedes Element zeigt auf einen nullterminierten String. Beispielsweise sind bei dem Aufruf

```
test Dies ist ein Beispiel
```

argc und argv wie folgt gesetzt:

```
argc hat den Wert 5
argv[0] zeigt auf das 't' von "test"
argv[1] zeigt auf das 'D' von "Dies"
argv[2] zeigt auf das 'i' von "ist"
argv[3] zeigt auf das 'e' von "ein"
argv[4] zeigt auf das 'B' von "Beispiel"
```

3. envp
Envp ist ein Array aus Environment-Strings. Das Ende wird durch einen Null-Pointer gekennzeichnet. Das Programm in *Abbildung 8*, demonstriert die Handhabung der Kommandozeilenparameter. Es gibt alle Parameter und anschließend alle Environmentvariablen mit ihren Belegungen aus. Die Ausgabe wird über printf() realisiert, das bereits in der letzten Folge besprochen wurde. Bitte beachten Sie, daß bei der Ausgabe von Strings das Formatzeichen %s verwendet wird. Die korrespondierende Variable ist dann vom Typ »Zeiger auf char«. Printf() erwartet also bei der Ausgabe von Strings eine Adresse. Beginnend bei dieser Adresse wird dann bis zum Nullbyte ausgegeben.

3.4 Parameter by reference
Ein wichtiger Anwendungsbereich von Pointern sind Funktionen mit Ausgangsparametern. Wir haben bereits am Beispiel von drawbox() gesehen, wie Funktionen mit Parametern versorgt werden und daß eine Funktion ein Ergebnis über das Return-Statement zurückliefern kann. Das Return-Statement kann aber nur zur Rückgabe elementarer Datentypen verwendet werden. Die Rückgabe von Arrays beispielsweise ist nicht möglich. Außerdem kann es vorkommen, daß eine Funktion mehr als nur ein Ergebnis an die aufrufende Funktion hochreichen muß.

Um dieses Problem zu lösen, übergibt die aufrufende Funktion nicht den Wert einer Variablen, sondern deren Adresse. Die aufgerufene Funktion kann dann an dieser Adresse Ergebnisse ablegen, also den Inhalt von Variablen der aufrufenden Funktion verändern. Wir wollen das Verfahren an einem Sourcecodefragment erläutern.

```
void fkt (int *pausgang)
{
    *pausgang=10;
}
```

erwartet die Adresse eines Integer als Parameter. Die Zuweisung *pausgang=10 schreibt den Wert 10 an diese Adresse. Wenn fkt() wie folgt aufgerufen wird:

```
void main()
{
    int   zahl;

    zahl=0;
    fkt (&zahl);
    printf ("wert hat jetzt den Wert %d \n!", zahl);
}
```

◄ *Abbildung 9: Typgenaue Dateneingabe.*

```
/* Typgenaue Dateneingabe */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "ckurs.h"

#define OKT          0
#define DEZ          1
#define HEX          2
#define CR          13
#define MAX         20
#define TRUE         1
#define FALSE        0
#define BACKSPACE    8          /* Code fuer Backspace      */
#define ESC         27          /* Code fuer Escape         */
#define BEEP        putch('\a') /* Piepton                  */

/* Prototypen                                                         */
void main (void);
int getvalue (int ze, int sp, int format, int laenge, long *wert);


int getvalue (int ze, int sp, int format, int laenge, long *wert)
{
        int     i;            /* Laufvariable            */
        int     ende;         /* Endeflag                */
        int     c;            /* Tasteneingabe           */
        char    s[MAX+1];     /* Buffer fuer Zahleneingabe */
        char    init[MAX+3];  /* Rahmen und Blanks       */

        static char *wertebereich[] =
        {
                "01234567",
                "0123456789",
                "0123456789ABCDEFabcdef"
        };

        if (laenge > MAX)
                laenge=MAX;

        *wert= 0l;

        /* Eingabefeld anzeigen */
        memset (init+1, ' ', laenge);
        init[0]    = '[';
        init[laenge+1]= ']';
        init[laenge+2]= '\0';

        ANSI_CURPOS (ze,sp-1);
        printf ("%s",init);
        ANSI_CURPOS (ze,sp);

        for (i=0, ende=FALSE; !ende; )
        {
                c = getch();

                /* Ziffer ? */
                if (i<laenge && strchr (wertebereich[format], c) != NULL)
                        putch (s[i++]=(char)c);
                else
                        /* Backspace ? */
                        if (c == BACKSPACE && i)
                        {
                                ANSI_CUR_LEFT;
                                putch (' ');
                                ANSI_CUR_LEFT;
                                i--;
                        }
                        else
                                /* Endetaste ? */
                                if (c == ESC || c == CR)
                                        ende = TRUE;
                                else
                                        /* Ungueltige Taste */
                                        BEEP;
        }

        /* Wenn Zeichen eingegeben wurden, dann konvertieren            */
        if (i)
        {
                s[i]='\0';
                switch (format)
                {
                        case OKT:
                                        sscanf (s, "%lo", wert);
                                        break;
                        case DEZ:
                                        sscanf (s, "%ld", wert);
                                        break;
                        case HEX:
                                        sscanf (s, "%lx", wert);
                                        break;
                }
        }

        return (c==CR ? TRUE : FALSE);
}

/*********************************************************************/

void main(void)
{
        long wert;

        ANSI_CLR();

        printf ("Dateneingabe Dezimalzahl mit Typ- und Laengenkontrolle.\n");
        printf ("\nEditieren  : Backspace"
                        "\nAbschicken : CR"
                        "\nAbbruch    : ESC");

        while (getvalue (8, 2, DEZ, 8, &wert))
        {
                ANSI_CURPOS (10,1);
                printf ("Eingabe ist %8ld !\n", wert);
        }
        ANSI_CLR();

}
```

hat Zahl nach Beendigung von fkt() den Wert 10. Es wurde eine Adresse übergeben (Adresse von zahl). Fkt() kann dann auf diese Adresse zugreifen.

C nimmt bei allen elementaren Datentypen eine Wertübergabe vor, d.h. es wird nur der aktuelle Wert einer Variablen auf den Stack gelegt. Die aufrufende Funktion arbeitet dann auf dieser Kopie, kann also den Wert des Originals in der aufrufenden Funktion nicht verändern.

Wir wollen das Call-by-Reference-Thema noch einmal anhand einer Funktion zur typgenauen Dateneingabe vertiefen. C bietet verschiedene Routinen, um Zeichen von der Standardeingabe (Tastatur) zu verarbeiten. All diese Funktionen (gets, getch, scanf) sind aber mit Nachteilen behaftet. So ist es beispielsweise nicht möglich, eine zeichenweise Typkontrolle bei der Eingabe vorzunehmen oder die Eingabelänge zu begrenzen.

Wir wollen eine Funktion entwickeln, die eine Eingabe von numerischen Werten in oktaler, dezimaler und hexadezimaler Notation gestattet. Die Länge eines Eingabefeldes soll über Parameter begrenzbar sein. Der Datentyp OKT, DEZ oder HEX wird ebenfalls als Parameter mitgereicht. *Abbildung 9* zeigt eine solche Funktion.

Besonders interessant ist hier die Verwendung des Ausgangsparameters wert. Hier steht nachher die eingegebene Zahl. Der Returnwert dient dazu, der aufrufenden Funktion mitzuteilen, ob die Eingabe ordnungsgemäß mit [Return] abgeschlossen (TRUE) oder mit [Esc] (FALSE) abgebrochen wurde. Getvalue() nutzt die Funktion getch(), um direkt ein Zeichen von der Konsole zu lesen. Zunächst wird abgeprüft, ob es sich um ein gültiges Zeichen handelt. Hierzu wurde die Variable wertebereich als Array aus Zeigern auf char definiert. Die einzelnen Strings enthalten die gültigen Zeichen. Über die Standardfunktion strchr() kann dann einfach überprüft werden, ob das eingegebene Zeichen gültig ist.

Als Sondertasten werden [Backspace], [Return] und [Esc] bearbeitet. Bei ungültigen Zeichen wird ein akustisches Signal ausgegeben.

Interessant ist auch die Funktion sscanf() (string scan formatted). Sie liest einen String gemäß einer Formatangabe, konvertiert die Zeichen und legt sie in den übergebenen Variablen ab. Sscanf ist eine Standardfunktion, die mit Ausgangsparametern arbeitet. An die entsprechenden Adressen überträgt sscanf() die formatierte Ergebnisse.

3.5 Operatoren

Bereits in der letzten Folge haben wir eine Reihe von Operatoren kennengelernt. Wir wollen das Thema Operatoren jetzt mit der Behandlung der Bitoperatoren abschließen. Zusätzlich werden die Prioritäten bei der Auswertung besprochen.

Bitoperatoren

Bitoperatoren arbeiten auf binären Zahlen. Eine Binärzahl besteht nur aus den Werten (0) und (1). Auch das Binärsystem ist ein Stellenwertsystem, bei dem die Stellung einer Ziffer innerhalb der Zahl über ihre Wertigkeit entscheidet, anders als beispielsweise im römischen Zahlensystem. Im Dezimalsystem beispielsweise, das auf der Basis 10 beruht, ist die Stelle n von der Wertigkeit 10 hoch n. Das Binärsystem basiert auf der Basis 2. Die einzelnen Stellen werden von rechts beginnend gezählt.

```
Der Wert der Dezimalzahl 123 ergibt sich als
Stelle 0: 3* (10 hoch 0)    3
Stelle 1: 2 *(10 hoch 1)   20
Stelle 2: 1 *(10 hoch 2)  100

Der Wert der Binärzahl 01111011 (Dezimal 123) ergibt sich als
Stelle 0: 1 *(2 hoch 0)    1
Stelle 1: 1 *(2 hoch 1)    2
Stelle 2: 0 *(2 hoch 2)    0
Stelle 3: 1 *(2 hoch 3)    8
Stelle 4: 1 *(2 hoch 4)   16
Stelle 5: 1 *(2 hoch 5)   32
Stelle 6: 1 *(2 hoch 6)   64
Stelle 7: 0 *(2 hoch 7)    0
```

Bitoperatoren arbeiten auf einer binären Zahlendarstellung. Die nun folgenden Bitoperatoren sind in C definiert.

Der UND-Operator (&)
In dem Ergebnis werden diejenigen Bits gesetzt, die in beiden Operanden gesetzt sind:

```
a      1010     (Dezimal: 10)
b      0111     (Dezimal: 7)
a&b    0010     (Dezimal: 2)
```

Der ODER Operator (|)
In dem Ergebnis werden diejenigen Bits gesetzt, die entweder in einem der beiden oder in beiden Operanden gesetzt sind:

```
a      1010     (Dezimal: 10)
b      0111     (Dezimal: 7)
a|b    1111     (Dezimal: 15)
```

Der EXKLUSIV-ODER Operator (^)
In dem Ergebnis werden diejenigen Bits gesetzt, die in einem aber nicht in beiden Operanden gesetzt sind. Beispiel:

```
a      1010     (Dezimal: 10)
b      0111     (Dezimal: 7)
a^b    1101     (Dezimal: 13)
```

Der KOMPLEMENT Operator (~)
Der Operator bezieht sich nur auf einen Operanden (unärer Operator). Die Bits werden invertiert:

```
a      1010     (Dezimal: 10)
~a     0101     (Dezimal: 5)
```

Die SHIFT-Operatoren (<<, >>)
Der linke Operand wird um so viele Positionen nach links (<<) oder rechts (>>) geshiftet, wie durch den rechten Operanden angegeben wird:

```
a      0101     (Dezimal 5)
a<<1   1010     (Dezimal 10)
a>>1   0010     (Dezimal 2)
```

Das Programm in *Abbildung 10* demonstriert die Handhabung der Bitoperatoren. Es liest zwei Operatoren von der Kommandozeile, verwandelt sie in numerische Werte und verknüpft diese.

```
#include <stdio.h>
#include <stdlib.h>
#include "ckurs.h"

/* Prototypen */
void main (int argc, char **argv);
void dispbin (unsigned short wert);

void main (int argc, char **argv)
{
        unsigned short  op1;
        unsigned short  op2;
        unsigned short  erg;

        if (argc != 3)
        {
                printf ("Aufruf %s Operand1 Operand2 !\n", argv[0]);
                exit (1);
        }

        sscanf (argv[1], "%d", &op1);
        sscanf (argv[2], "%d", &op2);

        printf ("\n%5u &  %5u = %5u ", op1, op2, erg = op1 & op2);
        dispbin (erg);
        printf ("\n%5u |  %5u = %5u ", op1, op2, erg = op1 | op2);
        dispbin (erg);
        printf ("\n%5u ^  %5u = %5u ", op1, op2, erg = op1 ^ op2);
        dispbin (erg);
        printf ("\n%5u << %5u = %5u ", op1, op2, erg = op1 << op2);
        dispbin (erg);
        printf ("\n%5u >> %5u = %5u ", op1, op2, erg = op1 >> op2);
        dispbin (erg);
        printf ("\n      ~  %5u = %5u ", op1, erg = ~op1);
        dispbin (erg);
}

void dispbin (unsigned short wert)
{
        register        i;

        printf ("Binär: ");
        for (i=0; i<16; i++, wert<<=1)
        {
                printf ("%c", wert & 0x8000L ? '1' : '0');
                if (i==7)
                        printf(" ");
        }
}
```

◂◂ *Abbildung 10: Bitoperatoren.*

Bitoperatoren sind unerläßlich, wenn ein bestimmtes Bit innerhalb eines Bytes oder Worts abgefragt werden soll, wie dies z.B. beim Interrupt 17h (INT 17h) benötigt wird. Der INT 17h des ROM-BIOS sendet ein Zeichen an den Drucker und liefert im Fehlerfalle den Druckerstatus in einem Register zurück. Der Status ist in folgendem Bitmuster kodiert:

```
Bit 7: 1 = Drucker bereit
Bit 6: 1 = Acknowledge
Bit 5: 1 = Kein Papier
Bit 4: 1 = Drucker Selektiert
Bit 3: 1 = Ein- Ausgabefehler
Bit 2:     wird nicht benutzt
Bit 1:     wird nicht benutzt
Bit 0: 1 = Time out
```

Den Fehler »Kein Papier« kann man wie folgt abfragen:

```
#define NO_PAPER        0x20

if (status & NO_PAPER)
   printf ("Kein Papier im Drucker !\n");
```

Das 5. Bit hat die Wertigkeit 32 (2 hoch 5). Bitmasken werden in der Regel in hexadezimaler Notation angegeben. Dezimal 32 entspricht hexadezimal 20 (0x20). Die Operation (status & NO_PAPER) setzt im Ergebnis nur Bits, die in beiden Operanden gesetzt sind. Wenn Bit 5 angeschaltet ist, liefert (status & NO_PAPER) den Wert 32, andernfalls 0. Da in C jeder Wert ungleich 0 als wahr gilt, wird in die IF-Anweisung verzweigt, wenn Bit 5 gesetzt ist.

3.6 Der Operator für bedingte Bewertung (?:) und der Kommaoperator (,).

C umfaßt zwei Operatoren, die etwas ungewöhnlich sind, aber keine großen Verständnisprobleme bereiten. Der Operator für bedingte Bewertung (?:) ist ein IF-ELSE-ähnliches Konstrukt auf Operatorebene. Der Anweisungsteil von dem ? wird ausgewertet. Liefert er einen Wert ungleich 0, wird der Anweisungsteil vor, andernfalls der hinter dem : ausgeführt. Die Anweisung

```
max= a > b ? a : b;
```

weist das Maximum von a und b an max zu. Es ist gleichbedeutend mit der IF-ELSE-Konstruktion:

```
if (a>b)
   max = a;
else
   max = b;
```

Der Kommaoperator , verbindet zwei Ausdrücke miteinander. Sie werden von links ausgewertet. Als Ergebnis des gesamten Ausdrucks gilt der Wert des rechten Operanden des Kommaoperators. Der Operator findet besonders in For-Schleifen Verwendung, um mehr als eine Laufvariable zu verwalten:

```
for (i=0, j=1; i<100 ; i++)
   /* ....*/
```

Häufig wird der Kommaoperator auch dazu verwendet, mehrere Variablen gleichen Datentyps zu deklarieren.

```
int   a, b, c;
```

deklariert drei Integervariablen a, b und c.

Eine weitere Verwendungsmöglichkeit ist die Kombination von Ausdrücken in einer IF-ELSE-Konstruktion. Hier kann man sich Klammern sparen:

```
if (argc < 3)
   printf ("Parameterfehler !\n"), exit(1);
```

3.7 Prioritäten und Reihenfolge der Auswertung

Die folgende Tabelle gibt die Prioritäten bei der Auswertung von Operatoren an.

	Operatoren	Zusammenfassung
1.	() [] -> .type	von links nach rechts
2.	! ~ ++ — + - * & (type) sizeof	von rechts nach links
3.	* / %	von links nach rechts
4.	+ -	von links nach rechts
5.	<< >>	von links nach rechts
6.	< <= > >=	von links nach rechts
7.	== !=	von links nach rechts
8.	&	von links nach rechts
9.	^	von links nach rechts
10.	|	von links nach rechts
11.	&&	von links nach rechts
12.	||	von links nach rechts
13.	?:	von links nach rechts
14.	= += -= *= /= %= &= ^= |= <<= >>=	von rechts nach links

Die Tabelle verzeichnet abnehmende Prioritäten. Die Operatoren der Stufe 1 haben die höchste Priorität. Die Operatoren + und - sind als Vorzeichenoperatoren mit nur einem Operanden (unärer Operator) und als binäre Operatoren vorhanden. Wenn sie als Vorzeichen benutzt werden, haben sie eine hohe Priorität und stehen auf Ebene 2.

Der Operator * hat ebenfalls eine Doppelbedeutung. Als unärer Operator dereferenziert er eine Adresse und als binärer Operator bewirkt er eine Multiplikation. Der Adreßoperator befindet sich auf Ebene 2, der Multiplikationsoperator auf Ebene 3.

Die Spalte »Zusammenfassung« gibt an, wie Operatoren zusammengefaßt werden, die sich auf einer Stufe befinden. Es gilt

```
a = 3;
```

Die Zuweisung

```
a += b = c = 2;
```

wird von rechts zusammengefaßt:

```
a += (b = (c = 2))    (a=5, b=2, c=2)
```

Die Sprache C definiert zwar den Vorrang der Operatoren bei der Auswertung von Ausdrücken, die zeitliche Auswertung eines Ausdrucks durch den Compiler ist aber nur für die Operatoren &&, ||, ?: und , definiert. Diese zeitliche Unbestimmtheit kann zu Problemen führen, wenn innerhalb einer Anweisung ein Ausdruck eine Variable ändert, auf die sich ein anderer Ausdruck bezieht:

```
x[i] = i++;
```

Es ist nicht sicher, ob der Index in den Array x schon erhöht wurde oder nicht. Der ++-Operator besagt zwar, daß das i auf der rechten Seite erst erhöht werden darf, nachdem es verarbeitet wurde. Nicht definiert ist aber, ob diese Erhöhung vor oder nach der Adreßberechnung auf der linken Seite erfolgen soll. Der Programmierer sollte derartige Seiteneffekte auf jeden Fall vermeiden. Die Ergebnisse sind compilerabhängig. Gleiches gilt auch bei Funktionsaufrufen. Für den folgenden Aufruf kann nicht gesagt werden, welche Werte die Parameter in der Funktion haben:

```
fkt (i, i++, i++)
```

Die Operatoren &&, ||, ?: und , nehmen eine Sonderstellung ein, da die zeitliche Auswertung dort immer von links nach rechts vorgenommen wird. Aus dieser zeitlichen Reihenfolge resultiert ein Effizienzgewinn bei der Auswertung von || und &&. Ausdrücke werden nur solange bewertet, bis ihr Wahrheitswert eindeutig feststeht. Beispiel:

```
if (a==5 || b==3)
```

Der linke Teil der Bedingung wird zuerst ausgewertet. Wenn a==5 zutrifft, steht der Wahrheitswert der Bedingung fest und der zweite Teil muß nicht mehr ausgewertet werden. Dies kann zu Problemen führen, wenn der rechte Operator eine Zuweisung enthält:

```
if (a==5 || (b= a+1))
```

b wird nur dann auf a+1 gesetzt, wenn a ungleich 5 ist.

In C liefert ein Ausdruck immer ein Ergebnis. Hierdurch sind komplexe Ausdrücke und Mehrfachzuweisungen wie

```
a=b=c=2*3;
```

möglich. Der Ausdruck c=2*3 liefert als Ergebnis den Wert von c (6), dieser wird an b zugewiesen, der schließlich an a.

Diese Flexibilität hat auch Schattenseiten. In der Abfrage

```
if (c=5)
    /*...*/
```

wurde aus dem Vergleichsoperator (==) ein Zuweisungsoperator (=). Der Ausdruck c=5 liefert den Wert 5. Da jeder Wert ungleich 0 als wahr gilt, verzweigt das Programm immer in das IF-Konstrukt. Ein solcher Fehler ist für C-Neulinge schwer zu finden.

4. Library Guide

In den Beispielen dieser Folge wurden die Memory- und die String-Funktionen, sowie die Funktionen der scanf-Familie angesprochen. Diese werden im folgenden erläutert.

Stringfunktionen
Strings sind in C als Array aus Charactern implementiert. Das Ende eines Strings wird durch ein sogenanntes Nullbyte (HEX 0) gekennzeichnet. Die Länge von Strings ist deshalb prinzipiell nicht begrenzt. Manche Compiler setzen aber eine Grenze bei 512 Bytes. Im folgenden werden alle ANSI-Stringfunktionen, die der MSC/QC-Compiler zur Verfügung stellt, kurz vorgestellt.
Schnittstellen

```
#include string.h

char *strcat (char *ziel, const char *quelle)
```

Hängt Zeichenkette quelle hinter Zeichenkette ziel.

```
char *strchr (char *string, int c);
```

Sucht von links nach dem Zeichen c in dem String string.

Spezialisten für Bürokommunikation und Anwendersoftware

Realisieren Sie die Anwendungen der 90er Jahre!



Hanseatische Software-
Entwicklungs- und
Consulting GmbH

Die HEC GmbH ist ein neues Unternehmen zur Lösung der bekannten gegenwärtigen und zukünftigen Aufgaben im Bereich integrierter Anwendersoftware. Ein kooperativer Führungsstil fördert Ihre Kompetenz und erwartet Ihre Mitverantwortung. Engagierte Mitarbeiter werden mit unseren ehrgeizigen Zielen mitwachsen.

Wir suchen Profis, die ihre Erfahrungen bei den großen Herstellern oder in innovativen Software- oder Systemhäusern erarbeitet haben.

Sie sollten möglichst übergreifende Erfahrungen in einem oder besser in mehreren Themenbereichen haben:

- Einführungsstrategien für Bürokommunikation
- Kommunikationsanalyse
- LAN-Operatingsysteme
- Presentation Manager als Applikationsumgebung für Anwendungsintegration
- OS/2, MS-DOS, UNIX
- SQL-fähige Datenbanken
- Retrievalsysteme
- ISDN-Anwendungen
- C-Realisierung
- Rationalität durch Industriestandards
- Systemeinführung und Schulung

Wenn Sie aus diesen Themenbereichen die Endusersoftware der 90er Jahre ableiten können, gehören Sie zu den Profis, die zu uns passen. Dann erkennen Sie auch Ihre Aktionsmöglichkeiten in einem startenden Unternehmen. Softwareentwickler und Berater sind aufgefordert!

Wir erwarten Ihre Bewerbungsunterlagen und freuen uns darauf, Sie kennenzulernen.

HEC GmbH
Geschäftsführung
Knochenhauerstr. 18/19
2800 Bremen 1

Telefon: 0421-30 90 60
Fax: 0421-309 06 99

Unsere Gesellschafter sind:

Das Bundesland Bremen
Siemens AG Berlin/München
pdv Beratungsgruppe

```
int strcmp  (const char *string1, const char *string2);
```

Vergleicht string1 und string2.

```
int strcmpi (const char *string1, const char *string2);
```

Vergleicht string1 und string2 ohne Berücksichtigung von Groß-/Kleinschreibung.

```
char *strcpy (char *ziel, const char *quelle);
```

Kopiert den String quelle auf die Adresse ziel.

```
size_t strcspn (const char *quelle, const char *zeichensatz);
```

Liefert den Index des ersten Zeichens in quelle, das in zeichensatz enthalten ist.

```
char *strdup (const char *quelle);
```

Reserviert Speicherplatz und legt dort eine Kopie von quelle ab.

```
size_t strlen (const char *string);
```

Ermittelt die Länge der Zeichenkette string.

```
char *strlwr(char *string);
```

Verwandelt alle Großbuchstaben aus string in Kleinbuchstaben.

```
char *strpbrk (const char *quelle, const char *zeichensatz);
```

Liefert einen Pointer auf das erste Zeichen in quelle, das in zeichensatz enthalten ist. Im Gegensatz zu strcspn() wird nicht der Index des Zeichens, sondern seine Adresse ermittelt.

```
char *strrchr (const char *quelle, int c);
```

Sucht von rechts nach dem Zeichen c in dem String string.

```
char *strrev (char *string);
```

Dreht die Zeichenkette string um.

```
char *strset (char *ziel, int zeichen);
```

Alle Zeichen von ziel werden auf zeichen gesetzt. Das Nullbyte bleibt erhalten.

```
char *strspn (const char *quelle, const char *zeichensatz);
```

Ermittelt den Index des ersten Zeichens in quelle, das nicht in Zeichensatz enthalten ist.

```
char *strstr (const char *string1, const char *string2);
```

Liefert den Pointer auf das erste Vorkommen von string2 in string1.

```
char * _strtime (char *time);
```

Setzt time auf die aktuelle Tageszeit.

```
char * _strdate (char *date);
```

Setzt date auf das Tagesdatum.

Die meisten Stringfunktionen liefern einen char-Pointer (char *) als Returnwert. Zwei Punkte sind hier besonders zu beachten:
1. Es ist wichtig, die Prototypen der Funktionen durch Einbinden von string.h bekanntzumachen. Andernfalls würde der Compiler davon ausgehen, daß die Funktionen einen INT-Wert liefern (Defaulteinstellung). Die Adresse eines Char ist aber etwas anderes als ein Integer. Unter MS-DOS ist ein Integer 2 Byte lang, ein Pointer auf ein Datenobjekt aber je nach gewähltem Speichermodell 2 (Small und Medium) oder 4 (Large und Huge) Bytes lang. Ist das Programm beispielsweise im Large-Model übersetzt und ist dem Compiler der Returnwert der Funktion nicht bekannt, so legt die Stringfunktion (z.B. strcat) zwar vier Bytes auf den Stack, die aufrufende Funktion holt aber nur zwei Bytes ab. Die Adresse ist verstümmelt.
2. Einige Stringfunktionen liefern die Adresse Ihres Zielobjektes als Returnwert. Hierdurch ist eine elegante Weiterverarbeitung in geschachtelten Funktionsaufrufen möglich. Durch

```
strcat (ziel, strcat (stamm, anhang));
```

wird der String anhang an den String stamm und das Ergebnis ziel gehängt. Der Zielstring muß lang genug sein, um den angehängten String aufzunehmen.

Wir wollen uns diesen Aspekt anhand eines kleinen Programms verdeutlichen:

```
main()
{
     char   name[23];
     char   vorname[16]

     strcpy(name,"Johann ");
     strcpy(vorname,"Wolfgang ");

     printf ("Der Bursche heißt: %s\n", strcat(name,
             strcat(vorname,"Goethe")));
}
```

Die Zeichenkette »Goethe« wird an den String »Wolfgang « angehängt. Die Zielvariable vorname ist lang genug, um beide Namen aufzunehmen (15 Bytes + Nullbyte). Strcat() liefert die Anfangsadresse des Zielstrings vorname als Returnwert. Dieser wird als Quellstring für den nächsten Funktionsaufruf verwendet.

Konstante Zeichenketten werden vom C-Compiler automatisch mit einem Nullbyte versehen und in einem bestimmten Programmsegment abgelegt. Dort wo die Zeichenkette verwendet wird, trägt der Compiler dann die Adresse innerhalb dieses Segmentes ein.

Ein umfangreiches Beispiel zur Stringverarbeitung finden Sie in *Abbildung 10*. Die Strings werden als Kommandozeilenparameter mitgegeben.

Memory-Funktionen
Die Standardbibliothek umfaßt Funktionen zur Verarbeitung beliebiger Speicherbereiche. Die Speicherbereiche müssen nicht wie bei den Stringfunktionen nullterminiert sein. Die Funktionen werden im folgenden beschrieben.
Schnittstellen

```
#include <memory.h>

void * memcpy (void * ziel, const void * quelle, size_t anzahl);
```

Kopiert anzahl Zeichen von quelle nach ziel.

```
int memcmp (const void *buffer1, const void *buffer2, anzahl);
```

Vergleicht die ersten anzahl Zeichen von buffer1 und buffer2.

```
void * memchr (void *buffer, size_t zeichen, unsigned anzahl);
```

Sucht in den ersten anzahl Zeichen von buffer nach dem Zeichen.

```
void * memset (void *ziel, int zeichen, size_t anzahl);
```

Setzt die ersten anzahl Zeichen von ziel auf zeichen.

Die Scanf-Funktionen
Die Scanf-Funktionen führen eine datentypspezifische, formatierte Eingabe durch. Sie sind das Gegenstück zur printf()-Funktionsfamilie.
Schnittstellen

```
#include <stdio.h>

int scanf (format [,argumente]...);
```

Scanf() interpretiert den Formatstring format und nimmt für jeden Platzhalter eine Eingabe des entsprechenden Typs entgegen. Für jeden Platzhalter innerhalb des Formatstrings muß die Adresse einer Variable des entsprechenden Typs angegeben werden. An diesen Adressen stellt scanf() dann die eingegebenen Zeichen ab.

Die Platzhalter werden wie bei printf() durch ein % eingeleitet und mit einem Datentyp abgeschlossen.

Scanf liest solange Zeichen von der Standardeingabe, bis ein »white space character« (Leerzeichen, Tab oder Newline) kommt, ein Zeichen angetroffen wird, das nicht dem einzulesenden Datentyp entspricht oder die optionale Eingabelänge erreicht ist. Alle folgenden Zeichen bleiben in der Standardeingabe und werden vom nächsten scanf() verarbeitet. Um die Standardeingabe zu leeren, kann der Funktionsaufruf fflush(stdin) benutzt werden.

Scanf liefert als Returnwert die Anzahl der eingelesenen Felder.

Scanf arbeitet ebenso wie printf() mit einer variablen Anzahl von Parametern. Der Programmierer muß dafür sorgen, daß die Anzahl der übergebenen Parameter mit der Anzahl der

```
/* Demonstriert die Handhabung bestimmter Stringfunktionen      */

#include <stdio.h>
#include <string.h>
#include <time.h>

#define VOKALE "aeiouAEIOU"

void main (int argc, char **argv);

void main (int argc, char **argv)
{
        char    buffer[200];
        char    *p;
        int     i;

        for (i=1; i<argc; i++)
              printf ("String %d: %-30s Länge %d \n", i, argv[i], strlen (argv[i]));

        for (buffer[0] = '\0', i=1; i<argc; i++)
                strcat (buffer, argv[i]);

     printf ("Stringverkettung mit strcat : %s Länge %d\n", buffer, strlen (buffer));
        printf ("Grossbuchstaben mit strupr  : %s \n", strupr(buffer));
        printf ("Kleinbuchstaben mit strlwr  : %s \n", strlwr(buffer));
        printf ("String rückwärts mit strrev : %s \n", strrev (buffer));

        printf ("\nHeute ist der %s\n", _strdate(buffer));
        printf ("Es ist jetzt %s Uhr \n", _strtime(buffer));

        for (p=argv[1], i=2; i<argc; i++)
                if (strcmp (p, argv[i]) >0)
                        p=argv[i];

        printf ("Der lexikalisch kleinste String : %s\n", p);

        printf ("\nFolgende Vokale sind enthalten :");
        for (i=1; i<argc; i++)
                for (p=argv[i]; (p=strpbrk(p, VOKALE)) != NULL; p++)
                        printf ("%c ", *p);
}
```

◄ *Abbildung 11: Stringfunktionen.*

Platzhalter übereinstimmt. Sind zuwenig Argumente angegeben, wird der nächste auf dem Stack liegende Wert als Adresse interpretiert und dort die Eingabe abgelegt. In der Regel werden hierdurch wichtige Informationen zerstört und das Programm verabschiedet sich.

Das Programm in *Abbildung 11* demonstriert die Handhabung von scanf(). Es wird mit einer Zeichenkette aus Formatierungszeichen aufgerufen und liest entsprechend dieser Formate die Datentypen ein. Nach jedem Schleifendurchlauf wird die Standardeingabe von Zeichen geleert.

```
int fscanf (FILE *stream, format [, argumente]);
```

Liest nicht von der Datei stream. Dateioperationen werden in einer späteren Folge besprochen.

```
int sscanf (const char *buffer, const char *format [,argumente]);
```

Liest den String buffer und konvertiert ihn gemäß den Formatbeschreibungen in format. Die Ergebnisse werden in den Argumenten des entsprechenden Datentyps abgelegt. Das Beispiel in *Abbildung 12* demonstriert anhand eines einfachen Taschenrechners das Einlesen über scanf() und die Formatierung über sscanf(). Das Beispiel macht sich die Tatsache zunutze, daß scanf() die Konvertierung eines Feldes abbricht sobald ein ungültiges Zeichen erscheint. Das Zeichen verbleibt im Eingabestrom und wird vom nächsten scanf() verarbeitet. Bei der Eingabe:

```
abb12 3 4 +
= 7.00
12 *
= 84.00
q
```

wird das q von dem scanf(), das den nächsten Operanden einlesen soll, im Eingabestrom belassen. Das folgende scanf() im Reinitialisierungsteil der For-Schleife verarbeitet es dann.

▶ *Abbildung 12: Ein einfacher Taschenrechner.*

```
#include <stdio.h>
#include <stdlib.h>

void main (int argc, char *argv[]);

void main (int argc, char *argv[])
{
        double  wert1;
        double  wert2;
        double  erg;
        char    operand;

        if (argc != 4)
        {
                printf ("\nAufruf: %s Zahl Zahl Operand !\n", argv[0]);
                printf ("\nBeispiel 3 4 +\n\n");
                exit (1);
        }
        printf ("Ende : q !\n");
        sscanf (argv[1], "%lf", &wert1);
        sscanf (argv[2], "%lf", &wert2);
        operand = argv[3][0];

        for (operand=argv[3][0]; operand != 'q'; scanf ("%c", &operand))
        {
                switch (operand)
                {
                        case '+':
                                erg = wert1 + wert2;
                                break;
                        case '-':
                                erg = wert1 - wert2;
                                break;
                        case '*':
                                erg = wert1 * wert2;
                                break;
                        case '/':
                                erg = wert1 / wert2;
                                break;

               }
               printf (" = %.2lf\n", erg);
               wert1 = erg;
               scanf (" %lf", &wert2);
        }
}
```

Das Beispiel verwendet die umgekehrte polnische Notation (UPN), die zuerst beide Operanden und dann den Operator erwartet.

```
int cscanf (format [,argumente]...);
```

liest direkt von der Konsole. Im Gegensatz zu scanf() ist eine Editierung der Eingabe über [Backspace] nicht möglich. Cscanf() terminiert auch sofort, wenn die Eingabelänge erreicht ist, ein abschließendes [Return] ist nicht erforderlich.

5. Getrennte Kompilierung

In dieser Folge haben wir zum ersten Mal ein Programm erstellt, daß aus zwei Sourcefiles bestand. In *Abbildung 5* ist die Funktion drawbox enthalten, in *Abbildung 6* die main()-Funktion. Im folgenden werden wir systematisch beschreiben, wie ein Programm aus mehreren Sourcefiles kompiliert und gelinkt wird. Wir gehen hierbei sowohl auf die QC-Entwicklungsumgebung ein als auch auf den MSC 5.1-Compiler.

Getrenntes Kompilieren und Linken mit dem MSC 5.1-Compiler

Um unter dem MSC 5.1 ein einzelnes Sourcefile zu kompilieren, muß der automatische Linker-Aufruf unterdrückt werden. Der Linker kann nämlich kein ausführbares File erzeugen, da die main()-Funktion fehlt. Der Aufruf

```
cl /W3 abb5.c
```

führt zu folgender Fehlermeldung des Linkers:

```
link: error L2029: unresolved externals:

_main in file(s):
c:\msc\lib\slibce.LIB(dos\crt0.asm)

There was 1 error detected
```

Der korrekte Aufruf unterdrückt den Linkeraufruf über die Option /c :

```
cl /W3 /c abb5.c
```

Es wird ein Objektfile mit dem Namen abb5.obj erzeugt. Anschließend muß das zweite File (abb6.c) mit

```
cl /W3 /c abb6.c
```

übersetzt werden. Der Linker kann dann beide Files zu einem EXE-File zusammenbinden:

```
link abb6+abb5,prog;
```

Es wird das Programm prog.exe aus den Objektdateien abb6.obj und abb5.obj erstellt.

Der Link-Aufruf ist wie folgt aufgebaut:

```
link [OPTIONEN] Objektdateien, [Name des EXE-Files],
     [Name des Map-Files], [Bibliotheken]
```

Optionen

Der Linker unterstützt zahlreiche Optionen, von denen hier nur die wichtigsten vorgestellt werden:

/MAP: Der Linker erstellt eine Mappingdatei mit einer Liste von öffentlichen Symbolen (Funktionsnamen und globale Variablen).

/NOI: Der Linker unterscheidet zwischen Groß- und Kleinbuchstaben. Ohne diese Option sind für den Linker die Funktionsnamen drawbox() und DrawBox() identisch.

/ST:number: Die Stackgröße wird auf number Bytes festgelegt. Die Default-Größe beträgt bei C-Programmen 2048 Bytes.

/CO: Der Linker schreibt symbolische Namen und Zeilennummern in das EXE-File, damit es von dem Codeview-Debugger verarbeitet werden kann. Dieses Flag sollte nur für Debugging-Zwecke benutzt werden. Das EXE-File wächst hierdurch um ca. 20-30 Prozent.

Objektdateien

Es muß mindestens eine Objektdatei angegeben werden. Mehrere Objektdateien werden durch + verbunden.

Name des EXE-Files

Hier kann der Name des ausführbaren Programms gewählt werden. Wenn die Angabe entfällt, nennt der Linker das Programm nach der ersten angegebenen Objektdatei.

Name des MAP-Files

Der Linker erstellt ein MAP-File, daß Informationen über Segmentgrößen enthält. Über die Option /MAP können zusätzlich die Namen aller öffentlichen Symbole in dieses File geschrieben werden.

Bibliotheken

Hier können sämtliche Bibliotheken angegeben werden, in denen der Linker nach Objektdateien suchen soll. Die Erstellung und Verwaltung von

Bibliotheken wird über den Library-Manager (lib.exe) vorgenommen. Er wird unten detailliert beschrieben.

Getrenntes Kompilieren und Linken in der QC-Entwicklungsumgebung
QC bietet eine sehr komfortable Verwaltung für umfangreiche Projekte in sogenannten »Program Lists«. Eine »Program List« kann die Namen von Bibliotheken, Objektdateien oder Sourcefiles enthalten. »Program Lists« haben die Default-Extension .MAK. Sie werden über den Menüpunkt »Make/Set Program List« erstellt. Die Menüpunkte »Make/Build Program« und »Make/Rebuild All« greifen auf die aktuelle »Program List« zu und erstellen das entsprechende EXE-File.

Wir wollen das Vorgehen im folgenden anhand der Sourcefiles abb5.c und abb6.c besprechen.

Wählen Sie den Menüpunkt Make/Set Program List und tragen Sie in das erste Feld den Namen des MAK-Files ein. Anschließend selektieren Sie die Dateien abb5.c und abb6.c aus dem Fenster »File List« und stellen Sie über die »Add/Delete« Operation in die »Program List« ein.

6. Das Utility-Programm Lib

Der Bibliotheksmanager LIB dient zur Verwaltung von Objektdateien in Bibliotheken. LIB kann die benötigten Informationen von der Kommandozeile, aus einer Responsedatei oder interaktiv einlesen. LIB unterscheidet zwischen Objektdateien und Modulen. Objektdateien können einen Pfadnamen und eine Extension haben, Module haben nur einen Namen.

Wenn LIB ohne Kommandozeilenargumente aufgerufen wird, erfragt er wie folgt die benötigten Angaben. Beispiel:

```
LIB

Library Name: ckurs.lib
Operations: +abb5
List file: ckurs.lst
Output Library:
```

Die fettgedruckten Angaben markieren die Eingaben des Benutzers.

Die Objektdatei abb5.obj wird in die Bibliothek ckurs.lib gestellt. Ein Verzeichnis aller in der Bibliothek enthaltenen Funktionen wird in die Datei ckurs.lst geschrieben. Als »Output Library« dient ebenfalls ckurs.lib. Die einzelnen Prompts werden im folgenden besprochen:

Library name: Es wird der Name einer existierenden Bibliothek erwartet. Ist die Bibliothek nicht vorhanden, legt LIB sie nach entsprechender Rückfrage an.

Operations: Ein Kommando besteht aus einem Kürzel (+,-,+-,* oder -*) gefolgt von einer Objektdatei. Sie können Objektdateien hinzufügen (+), Module löschen (-), austauschen (-+), in eine Datei kopieren (*) oder in eine Datei bewegen (-*).

List file: Das list file enthält ein cross-referencelisting mit allen öffentlichen Symbolen in einer Bibliothek und mit Verweisen auf die Module, in denen diese Symbole angesprochen werden.

Output Library: Wenn hier kein Bibliotheksname angegeben wird, nimmt LIB den unter »Library Name« spezifizierten Namen.

LIB kann auch benutzt werden, um mehrere Bibliotheken zusammenzufassen. Hierzu wird das +-Kommando im Feld »Operations« gefolgt von einem Bibliotheksnamen (Extension .lib) angegeben.

7. Aufgaben

7.1 Aufgabe 1

Schreiben Sie ein Programm, das einen Rahmen auf den Bildschirm zeichnet. Die Koordinaten und die Art des Rahmens sollen als Kommandozeilenparameter mitgegeben werden. Es gilt:

Linke obere Ecke: 1/1
Rahmenarten: Einfach == 1, Doppelt == 2
Beispiel:
Der Aufruf

```
aufgabel 1 1 20 80 2
```

zeichnet einen doppelten Rahmen von (1,1) bis (20,80).

7.2 Aufgabe 2

Erstellen Sie eine Funktion, die Lichtbalkenmenüs erzeugt. Die Funktion soll folgende Schnittstelle haben:

```
menu (int zeile, int spalte, char **texte, int anzahl, int start);
```

zeile: Startzeile des Menüs, spalte: Startspalte des Menüs, texte: Argv-ähnliche Struktur mit den einzelnen Menütexten, anzahl: Anzahl Menütexte, start: Startpunkt. Der Lichtbalken steht zuerst auf diesem Menüpunkt.

Der aktive Menüpunkt wird invers hervorgehoben. Die Funktion liefert die Nummer des ausgewählten Menüpunktes als Returnwert. Mit [Tab] wird der nächste mit [Shift][Tab] der vorherige Menüpunkt angewählt. [Return] wählt den aktiven Menüpunkt aus und [Esc] beendet das Menü ohne eine Auswahl.

Hinweis: Manche Tastenkombinationen (z.B. [Shift][Tab]) liefern einen erweiterten Code aus zwei Zeichen. Als erstes Zeichen wird eine 0 geschickt, anschließend der Scan-Code der Taste. Es empfiehlt sich, eine kleine Funktion zu schreiben, die alle Tastencodes auf *einen* numerischen Wert abbildet.

7.3 Aufgabe 3

Erstellen Sie einen Taschenrechner mit umgekehrter polnischer Notation (siehe *Abbildung 12*). Folgende Funktionen sind vorzusehen: Addition (+), Subtraktion (-), Multiplikation (*), Division (/), Shift links (<), Shift rechts (>), Bitweises UND (&), ODER (|), exklusives ODER (^) und Bit-Negation (~). Der Taschenrechner soll vier Zahlensysteme beherrschen: binäre (n), oktale (o), dezimale (d) und hexadezimale (n) Darstellung.

Hinweis: Diese Aufgabe ist etwas umfangreicher und schwieriger. Es ist empfehlenswert, eine eigene Eingabefunktion zu schreiben, die nur Ziffern des aktuell gültigen Zahlensystems zuläßt. Als Vorlage kann die Funktion aus *Abbildung 9* dienen. Als Datenmodell bietet sich ein Stack an, der die einzelnen Operanden enthält. Es sind zwei Operationen push() und pop() zu definieren, die Daten auf den Stack legen und von dort abholen.

8. Zusammenfassung und Ausblick

Dieser Seminarteil hatte die Schwerpunkte *Funktionen*, *Operatoren* und *einfache Pointer*.

Ein C-Programm kann sich aus einer beliebigen Anzahl von Funktionen zusammensetzen. Funktionen werden über Parameter mit Werten versorgt. C legt hierzu die Werte der aktuellen Parameter auf den Stack und die aufgerufene Funktion greift auf diese Werte zu. Dieses Verfahren bezeichnet man als Wertübergabe. Die aufgerufene Funktion arbeitet nur auf einer Wertkopie und hat keine Möglichkeit, die Originale zu verändern. Hierzu muß eine Adreßübergabe mit dem &-Operator erzwungen werden. C legt dann nicht den Wert einer Variablen, sondern deren Adresse auf den Stack. Die aufgerufene Funktion kann dann an diese Adresse schreiben und somit den Wert des Originals verändern.

Funktionen liefern in der Regel ein Ergebnis über das Return-Statement zurück. Der Datentyp dieses Ergebnisses ist entscheidend. Er muß bei der Deklaration angegeben werden. Als Defaulttyp gilt int.

Wenn ein Ergebnis nicht ausreicht, können mehrere Resultate über Ausgangsparameter übergeben werden. Beispielsweise stellt die Funktion scanf() die eingelesenen Werte über eine variable Anzahl von Ausgangsparametern zur Verfügung.

Pointer sind eine wichtige Datenstruktur in C. Ein Pointer ist eine Variable, die eine Adresse beinhaltet. Wichtig ist die Typbindung eines Pointers, d.h., der Datentyp auf den ein Pointer zeigt. Wird ein falscher Datentyp angegeben, führt eine Inkrementierung oder Dekrementierung des Pointers dazu, daß mit einer falschen »Schrittweite« durch den Speicher gelaufen wird.

C verfügt über eine große Anzahl von Operatoren. Es ist wichtig, zwischen der sachlichen Priorität und der zeitlichen Auswertungsreihenfolge zu unterscheiden. Die Priorität ist definiert. Die Auswertungsreihenfolge ist mit wenigen Ausnahmen nicht definiert. Bei den Operatoren || und && ist gewährleistet, daß zuerst die linke Seite ausgewertet wird, und die Auswertung abbricht, sobald der Wahrheitswert feststeht.

Der nächste Teil des C-Kurses wird sich eingehend mit den Dateioperationen in C beschäftigen. Wir werden, ausgehend von einfachen, alle in C angegebotenen Dateioperationen behandeln. Im Abschnitt Utilities werden wir uns den Sourcecode-Debuger CodeView ansehen und exemplarisch einige Beispiele debuggen, um uns die Darstellung der Variablen im Speicher anzusehen.

Jochen Witte, Rainer Kreutzer

Professionelle Techniken für die Anwendung von Strukturen und Unions in C-Programmen:

Mehr über Strukturen und Unions

Strukturen, Unions und Typedefs sind wichtige Mittel, um Daten in einem Programm zu strukturieren. Leider werden sie allzuoft nicht verstanden. In »Datenorganisation in C-Programmen« im Microsoft System Journal September/Oktober 1989, habe ich versucht, einige Ungereimtheiten im Umgang mit diesen Konstrukten zu klären. In diesem Artikel fahre ich fort mit Zeigern auf Strukturen, Zugriff auf Strukturen über mehrere Ebenen, Speicherallokierung und Arrays.

```
1  typedef struct {
2      s1      *s1ptr;
3      char    s1data[100];
4  } s1;
5
6  main()
7  {
8      s1   s1instance;
9
10     /* ... */
11 }
```

◂ *Bild 1: Eine Struktur mit einer Referenz auf sich selbst.*

```
1  typedef struct {
2      s2      *s2ptr;
3      char    s1data[100];
4  } s1;
5
6  typedef struct {
7      s1      *s1ptr;
8      char    s2data[100];
9  } s2;
10
11 main()
12 {
13     s1   s1instance;
14     s2   s2instance;
15
16     /* ... */
17 }
```

◂ *Bild 2: Zwei typedef-Anweisungen die sich gegenseitig referenzieren.*

Mein Artikel in der letzten Ausgabe sollte Sie überzeugt haben, öfters Typedef-Anweisungen in Ihren Programmen zu verwenden. Sie sollten sich aber ein Problem vor Augen halten. Strukturen, die eine Referenz auf sich selbst in Form eines Zeigers beinhalten, erzeugen mysteriöse Syntaxfehler. *Bild 1* zeigt einen solchen Fall. Das Problem ergibt sich aus der Tatsache, daß der Bezeichner s1 in Zeile 2 noch nicht abgeschlossen ist. Der Compiler kennt s1 zu diesem Zeitpunkt noch nicht und vermutet, daß ein ungültiger Typ verwendet wird.

Der im *Bild 2* gezeigte Fall ist eine andere Situation, aber ein ähnliches Problem. Zwei typedefs sind vorhanden, die aber jeweils auf den anderen verweisen (gegenseitige Abhängigkeit). Sie erhalten beim ersten typedef immer einen Fehler, da der zweite noch nicht existiert. Gibt es Lösungen zu diesen Problemen? Ja, und zwar einige. Ein paar sind falsch und ein paar sind verworren. Ohne klare syntaktische Vorstellungen versuchen viele Programmierer s1 * und/ oder s2 * in char * zu ändern (am besten beide, da die typedefs in der Regel in Includedateien enthalten sind, deren Reihenfolge variieren kann). Beim Zugriff oder der Zuweisung zu s1ptr oder s2ptr verwenden Sie dann einen Cast auf s1 * oder s2 *. Dies ist nicht ganz richtig und ein sicherer Weg für Schwierigkeiten auf einer anderen Hardware oder einem anderen Compiler. Der Cast nach s1 * oder s2 * ist ein sicheres Zeichen, daß früher oder später etwas fehlschlägt, da der Cast den Compiler abhält eine Fehlermeldung auszugeben (der Cast ist eine Direktive an den Compiler die ihm sagt, daß Sie wissen, was Sie mit einem falschen Typen tun. Es genügt momentan zu sagen, daß ein Cast, obwohl er Portabilität vorspiegelt, sie auf keine Weise garantiert).

► *Bild 3:*
Die Definition von Strukturen ohne typedef-Anweisung.

```
1   struct {
2       s2      *s2ptr;
3       char    s1data[100];
4   } s1;
5
6   struct {
7       s1      *s1ptr;
8       char    s2data[100];
9   } s2;
10
11  main()
12  {
13      s1   s1instance;
14      s2   s2instance;
15
16      /* ... */
17  }
```

►► *Bild 5:*
Unerlaubte formale Referenz.

```
1   struct atag {
2       struct anothertag  *anotherptr;
3       /* reference to another tag before
4          it comes into scope is fine */
5   };
6
7   struct sometag {
8       struct anothertag  anotherinstance;
9           /* This is an error since a
10             description of anothertag
11             is not in scope */
12  };
13
14  main()
15  {
16      /* ... */
17  }
```

Nachdem wir das Problem beschrieben haben, wollen wir uns jetzt Lösungsmöglichkeiten anschauen. Wir zerteilen das Problem in mehrere kleine Schritte. Auf dem Weg zur Lösung werde ich einige andere Konzepte erwähnen, die Sie möglicherweise noch nicht kennen. Sicher werden die Pascal-Anhänger glücklich sein, wenn ich sage, daß die Lösung »Vorwärtsreferenz« heißt. Wir werden uns einige Variationen dieser Technik ansehen.

► *Bild 4:*
Hinzufügen eines Strukturkennzeichen.

```
1   struct s1_tag {
2       struct s2_tag  *s2ptr;
3       char    s1data[100];
4   };
5
6   struct s2_tag {
7       struct s1_tag  *s1ptr;
8       char    s2data[100];
9   };
10
11  main()
12  {
13      struct s1_tag s1instance;
14      struct s2_tag s2instance;
15
16      /* ... */
17  }
```

Vorwärtsreferenzen

Gestalten wir das Problem vorerst etwas einfacher, indem wir die typedefs aus dem Beispiel entfernen (*Bild 3*). Es sollte jetzt klar sein, daß es keine Möglichkeit gibt, einen Zeiger auf s1 oder s2 im Beispielprogramm zu referenzieren. Anders ausgedrückt, s1 und s2 sind keine Typen. Wie sieht es aber aus, wenn wir jeder Deklaration ein Strukturkennzeichen hinzufügen, so daß die Struktur referenziert werden kann (*Bild 4*)? Jetzt beziehen sich s2ptr und s1ptr nicht auf einen Typ, sondern sie dienen nur als Referenz auf eine Struktur.

Bedenken Sie aber auch, daß eine Vorwärtsreferenz in Form einer Zeiger-Deklaration zwar zulässig ist, die Struktur selbst aber nicht verwendet werden darf. Im *Bild 5* funktioniert die Deklaration von anotherptr, da der Zeiger das Aussehen der Struktur (anothertag), auf die er zeigt noch nicht kennen muß. Dies ist ein Weg in C, wie ein unvollständiger Typ vorkommen kann. Wenn Sie aber anotherptr einen Wert zuweisen wollen, müssen Sie zuerst die Struktur definieren, auf die er zeigen soll.

Genauso ist die Deklaration von anotherinstance falsch, da ich versucht habe, in sometag eine ganze Struktur anothertag zu deklarieren. Dies kann nicht funktionieren, da der Compiler die Mitglieder von sometag nicht kennt, und daher die Größe nicht bestimmen kann. Der Compiler kann die Größenbestimmung von sometag nicht verzögern, bis die Größe von anothertag bekannt ist, und muß deshalb eine Fehlermeldung ausgeben.

Das Problem nicht kompletter Typen in C ist wichtig, und man muß es sich ständig vor Augen halten. Es geht noch weit über die hier diskutierten Fälle hinaus, und bleibt nicht auf Strukturen beschränkt. In der Regel können Sie einen nicht komplett definierten Typen immer referenzieren, aber ihn nie als ganze Einheit verwenden. Eine andere Stelle, an der nicht komplette Typen häufig verwendet werden, ist die Deklaration von externen Arrays, wie bei extern int array[]. Sie können hier indizieren, oder sich die Adresse des Arrays geben lassen, aber die Operation sizeof(array) nicht durchführen, da die Größe des Arrays in der Quelldatei, in der diese Deklaration steht, nicht bekannt ist. Arrays dienen nur als Deklarationen, nicht als Definitionen.

Typedef- und Strukturkennzeichen

Wenn Sie überlegen #define oder typedef zu verwenden, so ist gewöhnlich typedef die bessere Wahl. Ich habe noch eine weitere Quelle für Verwirrung eingeführt, indem ich Strukturen und Strukturkennzeichen verwendet habe. Es sollte aber nicht der falsche Eindruck entstehen, daß ich meine Strukturkennzeichen seien besser als typedefs, da dies nicht unbedingt der Fall ist. Ich habe die Tags nur als Sprungbrett verwendet.

Da jetzt die Vorwärtsdeklarationen klarer sind, kehren wir zum Originalproblem zurück, und lösen es mit typedef. Es gibt zwei gebräuchliche Wege, dies zu tun (*Bild 6* und *Bild 7*). Beide sind gleichbedeutend. Im ersten Beispiel werden zwei typedef-Anweisungen verwendet, die Strukturkennzeichen referenzieren (Da ein typedef nur als Synonym dient, handelt es sich nur um ein Strukturkennzeichen, das später definiert werden kann). Jetzt sind die beiden

◂◂ *Bild 6: Typedef-Anweisungen, die Strukturkennzeichen referenzieren.*

```
1   typedef struct s2_tag s2;
2   typedef struct s1_tag s1;
3
4   struct s1_tag {
5       s2  *s2ptr;
6       char    s1data[100];
7   };
8
9   struct s2_tag {
10      s1  *s1ptr;
11      char    s2data[100];
12  };
13
14  main()
15  {
16      struct s1_tag s1instance;
17      struct s2_tag s2instance;
18
19      /* ... */
20  }
```

◂◂ *Bild 7: Typedef-Anweisungen, die intern Strukturkennzeichen verwenden.*

```
1   typedef struct s1_tag {
2       struct s2_tag  *s2ptr;
3       char    s1data[100];
4   };
5
6   typedef struct s2_tag {
7       struct s1_tag  *s1ptr;
8       char    s2data[100];
9   };
10
11  main()
12  {
13      struct s1_tag s1instance;
14      struct s2_tag s2instance;
15
16      /* ... */
17  }
```

typedefs gültig, und sie können verwendet werden, indem die Namen s1 und s2 wie Typen eingesetzt werden. Dies sogar dann, wenn sie in der Strukturdefinition verwendet werden, auf der sie basieren, wie dies im Bild 6 gezeigt wird.

Die zweite Methode (*Bild 7*) liefert dasselbe Ergebnis, aber mit einer anderen Methode. Es ist eigentlich die Umkehrung der vorherigen Vorgehensweise. Anstatt typedef für die Namen zu verwenden und Strukturkennzeichen zu erzeugen, erzeugen wir die Strukturkennzeichen während wir die typedef-Anweisungen verwenden.

Strukturen übergeben und zurückerhalten

Ich nehme an, der Leser weiß, wie er Strukturen an Funktionen übergibt. Es genügt, zu sagen, daß Sie immer einen Zeiger auf die Struktur oder die Adresse der Struktur übergeben sollten (&struct_var, was ein Zeiger auf die Struktur ist), anstatt die gesamte Struktur selbst. Die Struktur sollte als Call by Reference nicht als Call by Value übergeben werden. Der zweite Fall kann viel Stackspeicher benötigen und sporadische Fehler verursachen, wenn Sie nicht vorsichtig sind.

Die Rückgabe einer Struktur sollte aus Effizienzgründen auch durch einen Zeiger auf die Struktur geschehen. Es gilt drei Fälle zu unterscheiden: Die Rückgabe einer kompletten Struktur, die Rückgabe eines Zeigers auf eine Struktur, und den Fall, bei dem nur einige Funktionsmitglieder benötigt werden, nachdem die Funktion diese zurückgegeben hat.

◂ *Bild 8: Die Rückgabe einer kompletten Struktur von einer Funktion.*

```
1   struct ps1 {
2       char    *p;
3       char    array[1024 - sizeof(char *)];
4   } v1;
5
6   struct ps2 {
7       char    *p;
8       char    array[512 - sizeof(char *)];
9   } v2;
10
11  struct ps1 f1()
12  {
13      v1.p = "hi there";
14      return (v1);
15  }
16
17  struct ps2 f2()
18  {
19      v2.p = "HI THERE";
20      return (v2);
21  }
22
23  main()
24  {
25      func(f1(), f2());
26      func2(&f1(), &f2()); /* This doesn't mean the
27                              address of f1 and f2!
28                              It means the address
29                              of the structures they
30                              return—Remember
31                              precedence! */
32  }
33
34  func(struct ps1 v1, struct ps2 v2)
35
36
37  {
38      printf("%s\n", v1.p);
39      printf("%s\n", v2.p);
40  }
41  func2(struct ps1 *v1, struct ps2 *v2)
42
43
44  {
45      printf("%ld\n", v1); /* print out the addrs
46                              that v1 and v2 point
47                              to, these */
48      printf("%ld\n", v2); /* may also be printed
49                              with a %p instead of
50                              %ld */
51      printf("%s\n", v1->p);
52      printf("%s\n", v2->p);
53  }
```

Rückgabe einer Struktur

Wenn Sie eine komplette Struktur als Funktionsergebnis zurückgeben, sollten Sie daran denken, was der Compiler in diesem Fall tut. Er muß die komplette Struktur in einen anderen Speicherbereich kopieren. Dieser Bereich könnte zum Beispiel eine Union aller Strukturen Ihres Programms sein. Dieser Bereich wäre dann groß genug, um jede Struktur aufnehmen zu können. Dieser Bereich muß ohne spezielle Eigenarten adressierbar, statisch und korrekt ausgerichtet sein.

Dieses Kopieren wird in der Regel von einer Standardfunktion des Compilers erledigt. Es gibt aber auch Probleme. Sehen Sie sich das Programm in *Bild 8* und speziell den Aufruf von func in Zeile 25 an. Dies arbeitet genauso, als ob f1 und f2 Basistypen zurückliefern, da auch Strukturen als Funktionsergebnis zulässig sind. Führen wir uns die Reihenfolge der Ereignisse vor Augen. F1 liefert eine Struktur, die in den statischen Bereich kopiert wird. Anschließend wird der statische Bereich auf den Stack kopiert, was bei einigen Systemen eine Menge »Move Long Word«-Befehle erzeugt, statt einer Schleife. Dasselbe geschieht mit dem Ergebnis von f2.

▶ *Bild 9: Die Rückgabe eines Zeigers auf eine automatische Struktur.*

```
 1  struct temp {
 2      int    members;
 3      /* ... */
 4  };
 5
 6  struct temp *
 7  func()
 8  {
 9      struct temp temp; /* yes, structure tags *and*
10                           structures can have the
11                           same name */
12
13      return (&temp);
14  }
15
16  main()
17  {
18      struct temp *temp;
19
20      temp = func();
21      /* <expressions using temp->???> */
22  }
23
```

Die Folge ist ein langsames Programm. Aber das ist nur die eine Hälfte. Übergeben Sie die Adressen der Strukturen, die von f1 und f2 geliefert werden, wie dies in Zeile 26 der Fall ist (Sie können dies tun, da eine Struktur ein Verbundtyp ist), so werden Sie sehen, daß Ihr Compiler nicht in der erwarteten Art und Weise arbeitet. F1 und f2 können dieselbe Adresse zurückgeben, weshalb die Adressen und Strings, die die Funktion func2 ausgibt dieselben sein können.

Rückgabe eines Zeigers

Der zweite Fall ist einfach. Geben Sie keinen Zeiger auf eine automatische Struktur zurück (*Bild 9* zeigt ein Beispiel dieses Fehlers). Func gibt &temp zurück, was in C erlaubt ist, aber nicht richtig funktioniert. Wenn func zurückkehrt, ist die temp-Struktur, nicht das Tag temp, das Gültigkeit in der ganzen Datei besitzt, nicht mehr verfügbar, da sie nur in der Funktion gültig ist. Da temp nicht mehr verfügbar ist, ist der Zugriff undefiniert, und kann sogar zum Programmabsturz führen. Ändern der Deklaration in static struct *temp temp;* verursacht keine Probleme. Da temp jetzt statisch ist, ist die Struktur während der Lebensdauer des Programms immer gültig. Es ist gleichgültig, ob die Funktion, in der temp deklariert ist, ausgeführt wird, oder nicht. Wichtig ist, daß temp verfügbar ist, da es statisch ist.

Es spielt auch keine Rolle, ob temp gültig ist oder nicht. Gültigkeitsbereiche beziehen sich nur auf Bezeichner. Es berührt nicht die Konzepte von Variablen oder Adressen. Sie erhalten selbstverständlich die Adresse eines Bezeichners nur, wenn dieser gültig ist. Wenn Sie aber die Adresse einmal während seiner Lebensdauer erhalten haben, können Sie mit ihr tun was Sie wollen (auch fehlerhaftes).

Zugriff auf Elemente

Der letzte Fall ist der Zugriff auf einzelne Elemente einer Struktur, die von einer Funktion zurückgegeben wird. Denken Sie einen Augenblick darüber nach. Sollten Sie auf die Elemente nach der Rückgabe der Struktur mit dem ganzen Kopieren zugreifen, oder sollten Sie sich nur einen Zeiger auf die Struktur geben lassen?

Im Falle eines Betriebssystem-Aufrufs werden Sie keinen Zeiger auf die internen Strukturen des Betriebssystems wollen, es sei denn, Sie schreiben einen Gerätetreiber oder eine Applikation, die spezielles Wissen voraussetzt.

In dieser Situation kann es Ihnen passieren, daß Sie eine C-Bibliotheksfunktion aufrufen, statt einen Systemaufruf durchführen und auf etwas im Speicherbereich Ihres Prozesses zugreifen. Aber die Argumente zu dieser Bibliothek können und sollten nicht verändert werden. Routinen können ein Argument, das ein Zeiger auf eine Struktur ist, akzeptieren (eine die Sie auf Grund einer Include-Datei richtig allokiert haben) und keinen Wert zurückgeben, der eine Struktur darstellt.

Die Routine kann die Strukturmitglieder selbst initialisieren. Dies hilft, die vorher beschriebene Situation zu vermeiden, und ist eine gute Möglichkeit, einige Ihrer Programme zu schreiben. Zusätzlich können Sie Routinen finden, die große Strukturen benutzen, und anstatt diese ganz zu übergeben, erzeugen sie eine kleinere Struktur, die die benötigte Untermenge enthält, und manipulieren diese.

Abhängig von Ihrer Programmlogik kann es auch ratsam erscheinen einen Zeiger auf eine Struktur zurückzugeben, auch wenn nur ein Element benötigt wird. Sie brauchen nur den Pfeiloperator (->) für den zurückgelieferten Zeiger, um auf das gewünschte Element zuzugreifen, anstatt mit der gesamten Struktur zu arbeiten. Erinnern Sie sich, daß Sie eine Funktion aufrufen, die ein statisches Vorkommen der Struktur beherbergt, auf die sie einen Zeiger zurückgibt. Die Folge ist aber, daß der zweimalige Aufruf der Funktion die Werte des vorherigen Aufrufs zerstören kann. Die aufrufende Funktion muß dies in Betracht ziehen, und alle Werte, die sie benötigt, kopieren, bevor die Funktion erneut aufgerufen wird.

Dies führt zur nächsten Situation, in der entweder der Aufrufer oder der Aufgerufene die Struktur dynamisch allokiert. Sie müssen dann sorgfältig die Allokierung der Struktur kontrollieren. Ebenso sorgfältig sollte das Freigeben vor sich gehen.

Strukturzugriff

Bild 10 zeigt sowohl Strukturen mit Zeigern auf andere Strukturen als auch Vorkommen der anderen Strukturen. Es besteht die Möglichkeit, daß Sie dies nie benötigen. Ich zeige es aber trotzdem, da es neue Erkenntnisse über Strukturen liefert.

```
 1  struct s1tag {
 2      int    s1var;
 3  } s1;
 4
 5  struct s2tag {
 6      int    s2var;
 7      struct s1tag *ps1;
 8      struct s1tag s1inst;
 9  } s2;
10
11  struct s3tag {
12      int    s3var;
13      struct s2tag *ps2;
14      struct s2tag s2inst;
15  } s3;
16
17  struct s3tag *ps3;
18
19  main()
20  {
21      s1.s1var = 99;
22
23      s3.s2inst.s1inst.s1var = 5;
24
25      ps3 = &s3;
26      ps3->ps2 = &s2;
27      ps3->ps2->ps1 = &s1;
28      ps3->ps2->ps1->s1var = -99;
29      printf("s1var=%d\n", s1.s1var);
30
31      /* ps3->ps2.s1inst.s1var = 11; */
32      ps3->ps2->s1inst.s1var = 22;
33      /* ps3->(*ps2).s1inst.s1var = 33; */
34      /* ps3->*ps2.s1inst.s1var = 44; */
35      /* ps2 = 0; */
36        /* might as well have called this 'abccba' */
37      (*ps3->ps2).s1inst.s1var = 55;
38      /* *ps3->ps2.s1inst.s1var = 66; */
39      printf("s3.s2inst.s1inst.s1var=%d\n",
40                s3.s2inst.s1inst.s1var);
41
42      ps3->ps2 = &ps3->s2inst;
43      ps3->ps2->s1inst.s1var = 22;
44      printf("s3.s2inst.s1inst.s1var=%d\n",
                  s3.s2inst.s1inst.s1var);
45  }
```

◄◄ *Bild 10: Zugriff über mehrere Ebenen.*

Die meisten von Ihnen können ohne Zweifel eine einfache Strukturreferenz, wie in Zeile 21 von Bild 10 erzeugen. Viele von Ihnen können auch einen Zugriff über mehrere Ebenen von Strukturen wie in Zeile 23 erzeugen, aber mit der Unsicherheit, ob es richtig ist (es ist). Alles darüber hinaus (auch diese unschönen Gebilde, die wir Zeiger nennen) ist jedoch ungewiß.

Um Zeile 23 zu verstehen, müssen Sie wissen, daß s3 die Struktur s2tag beinhaltet, die s2inst heißt. Diese wiederum enthält die Struktur s1tag (s1inst), die eine Integerzahl s1var enthält.

Um jede dieser Vorkommen zu benutzen, brauchen sie einfach einen Strukturzugriff für jedes Element, wie Sie es in Zeile 21 sehen. Da der Vorrang des Punkt-Operators (.) keine Probleme bereitet, und seine Assoziativität natürlich von links nach rechts ist, ist der Zugriff einfach,

```
Struktur1.<...>.StrukturN
```

wie in Zeile 23 gezeigt.

Der gesamte Platz ist in s3 vorhanden, da s3 eine Struktur s2tag und s2tag eine Struktur s1tag enthält. Im anderen Fall, wie in den Zeilen 25-28 zu sehen ist, sind von ps3 Referenzen auf s3tag, s2tag und s1tag durch die Zeiger ps2 und ps1 enthalten, die Zugriff auf andere Variable wie s1 und s2 außerhalb von s3 erlauben.

In Zeile 25 wird ps3 mit der Adresse von s3 initialisiert. Wir müssen hier &s3 statt s3 verwenden, da wir die Adresse von s3 benötigen, und dem Zeiger nicht den Inhalt der Struktur s3 zuweisen wollen. Danach können wir auf die Elemente der Struktur (in diesem Falle s3) zugreifen, auf die ps3 zeigt, indem wir den Operator -> verwenden. Wir führen eine Wertzuweisung für einen anderen Strukturzeiger (Zeiger auf s2tag) ps2 durch. Diese Zuweisung ist identisch derjenigen von ps3.

Die Auswertung des Pfeil-Operators wird in genau der Weise wie beim Punkt-Operator durchgeführt. Auch hier liegt eine Bindung von links nach rechts vor. Das Verständnis von Zeile 23 verbunden mit der Diskussion gestaltet die Interpretation der Zeilen 27 und 28 einfach. Zeile 27 benutzt die Tatsache, daß ps3->ps2 s2tag referenziert, welche eine Referenz auf s1tag enthält. Dies gestattet die Zuweisung von ps3->ps2->ps1 an einen Bezeichner wie s1, der die Größe und Gestalt von s1tag besitzt. Zeile 28 verwendet ps3->ps2->ps1, was einen Zeiger auf s1tag darstellt. Hier ist dann eine Referenz auf ein Element, wie zum Beispiel s1var möglich.

Jeder Zeiger, der in den Zeilen 25-27 verwendet wird, muß initialisiert werden. Sie können nicht einfach den Zugriff auf s1var in Zeile 28 codieren, ohne die anderen Zuweisungen durchzuführen. Dies wäre falsch, da jeder Zeiger auf den richtigen Speicherbereich zeigen muß, bevor er verwendet wird.

Seien Sie vorsichtig, da Sie dies als Programmierer beachten müssen. Den Compiler kümmert es nicht, wenn Sie nur Zeile 28 ohne die Zeilen 25 bis 27 kodieren. Sie können die Strukturzeiger-Zuweisungen auch in einem anderen Teil des Programms in einer If/Else-Logik codiert haben, und dies nicht unmittelbar vor Zeile 28 durchführen. Der Compiler kann es einfach nicht kontrollieren.

Die Moral von der Geschichte ist, daß Sie alle Zuweisungen sauber codieren sollten und sich vergewissern, daß die Zeiger vor dem Verwenden richtig initialisiert sind, da der Compiler Ihnen keine Warnung oder Fehlermeldung für eine unterlassene Initialisierung geben kann. Probleme dieser Art treten zur Laufzeit des Programms auf. Oftmals sind sie sporadisch und nur sehr schwer zu finden. Wenn Sie die zusätzlich nötige Sorgfalt beim Codieren walten lassen, können Sie viele dieser Situationen vermeiden.

Zeile 29 bedeutet weniger, als Sie vermuten werden. Wenn Sie das Programm näher ansehen, werden Sie feststellen, daß die Zeilen 27 und 28 zum Zugriff auf s1.s1var dienen. Dies funktioniert auch, wenn die Zeiger ps3 und/oder ps3->ps2 nicht gültig sind. Ich sollte sagen, dies scheint richtig zu funktionieren – sehen Sie das Problem?

Nehmen Sie an, daß Zeile 25 versehentlich aus der Quellcode-Datei gelöscht wurde. Arbeitet das Programm dann noch richtig? Vielleicht. Wenn es richtig arbeitet, ist dies korrekt? Nein. Wie oben erwähnt, wird es fehlerfrei übersetzt. Es ar-

▸ *Bild 11: Zugriff über mehrere Ebenen mit Funktionen.*

```
 1  struct s1tag {
 2      int    s1var;
 3  } s1 = { 12345 };
 4
 5  struct s2tag {
 6      int    s2var;
 7      struct s1tag *ps1;
 8      struct s1tag s1inst;
 9  } s2;
10
11  struct s2tag *ps2;
12
13  struct s1tag
14  s1returner()
15  {
16      return (s1);
17  }
18
19  struct s1tag *
20  ps1returner()
21  {
22      return (&s1);
23  }
24
25  void example1()
26  {
27      printf("%d\n", s1.s1var);
28      printf("%d\n", s1returner().s1var);
29      printf("%d\n", ps1returner()->s1var);
30      printf("%d\n", (*ps1returner()).s1var);
31      printf("%d\n", (ps1returner())->s1var);
32      /* printf("%d\n", (ps1returner()).s1var); */
33  }
34
35  void example2()
36  {
37      struct s1tag s1holder;
38      struct s1tag *ps1holder;
39
40      printf("%d\n", s1.s1var);
41      s1holder = s1returner();
42      printf("%d\n", s1holder.s1var);
43      ps1holder = ps1returner();
44      printf("%d\n", ps1holder->s1var);
45      printf("%d\n", (*ps1holder).s1var);
46  }
47
48  struct s2tag
49  s2returner()
50  {
51      return (s2);
52  }
53
54  struct s2tag *
55  ps2returner()
56  {
57      return (&s2);
58  }
59
60  void example3()
61  {
62      printf("%d\n", s2.s1inst.s1var);
63      printf("%d\n", s2returner().s1inst.s1var);
64      printf("%d\n", ps2returner()->s1inst.s1var);
65  }
66
67  void example4()
68  {
69      printf("%d\n", ps2returner()->ps1->s1var);
70  }
71
72  main()
73  {
74      example1();
75      example2();
76
77      s2.s1inst.s1var = 54321;
78      example3();
79
80      s2.ps1 = &s1;
81      example4();
82  }
```

beitet unter DOS, als ob alles in Ordnung ist. Die Betriebssysteme Xenix und OS/2 erzeugen aber einen generellen Schutzverletzungsfehler, da der Speicherzugriff nicht erlaubt ist.

Ps3 ist eine externe Variable (das heißt in der selben Datei extern definiert mit keiner Initialisierung), und wird deshalb implizit mit 0 initialisiert. In diesem Fall wird ps2 aus einem Zeiger gebildet, der auf die Speicherzelle 0 zeigt. Die Zuweisung schreibt dann etwas an die Speicherzelle 0 + sizeof(int), was in der Regel die Speicherzelle 2 ist. Dieser Wert wird, was, wie wir alle wissen, falsch ist, als ps3->ps2 angenommen. Wenn ein Zugriff auf diese Speicherzelle in diesem Augenblick keinen Schaden anrichtet, so wird das Programm ohne erkennbare Fehler weiter ausgeführt.

Der Microsoft-C-Compiler ab der Version 5.0 schützt normalerweise vor diesem Fall mit der bekannten Laufzeit-Fehlermeldung R6001 nach dem Programmlaufende. Sie sollten sich aber nicht darauf verlassen. Diese Fehlermeldung tritt nur auf, wenn Sie in den unteren Speicherbereich schreiben. An einer anderen Stelle weist Sie niemand darauf hin. Sorgen Sie also dafür, daß Ihre Zeiger richtig initialisiert sind.

Sorgfältige Blicke helfen in diesem Fall. Ein gutes Beispiel ist, wenn nach dem Zusammenfügen einiger Programmteile das Programm verrückt spielt. Dies wird von einem Zeiger verursacht, der auf eine Stelle zeigt, die nicht verändert werden darf, ohne Probleme zu verursachen. Dies ereignet sich, da Ihre Programmodule sich in unterschiedlichen Speicherbereichen befinden. So zeigen sich dann Fehler, die bei den einzelnen Programmteilen nie aufgefallen sind.

Mit diesen Informationen können wir uns den Rest von Bild 10 ansehen. Wollen wir s1inst indirekt durch den Zeiger ps2 ansprechen, so können wir ein Statement wie in Zeile 31 verwenden. Lesen wir das von links nach rechts (da beide Operatoren -> und . die gleiche Priorität haben und von links nach rechts binden), so sehen Sie, daß der Teil, der ps2.s1inst referenziert, fehlerhaft ist. Da ps2 ein Zeiger ist, funktioniert alles mit ps2. (beachten Sie den Punkt) nicht, da der Punkt-Operator voraussetzt, daß der linke Operand ein Strukturtyp ist. Dieser Strukturtyp muß entweder ein Bezeichner sein, der einen Strukturnamen darstellt, oder eine Dereferenzierung eines Zeigers auf eine Struktur wie bei ps = s; ... (*ps).m...;

Den richtigen Weg sehen Sie in Zeile 32, die ähnlich der Zeile 27 ist, nur greifen wir ohne die Dereferenzierung eines weiteren Zeigers auf eine Struktur (s1list) zu.

Sie sollten eine andere Lösung dieses Problems erkennen, da Sie durch Ihre C-Kenntnisse wissen, daß eine Strukturreferenz wie Zeiger->Element in das Konstrukt (*Zeiger).Element übertragen wird. Wie können wir das verwenden? Zeile 33 zeigt eine gute Möglichkeit, dieses Problem zu lösen, aber es geht nicht weit genug. Unglücklicherweise bringt Ihnen das Übersetzen nichts, da die meisten Compiler einfach einen Syntaxfehler melden. Das ist auf den ersten Blick unverständlich.

Sie müssen sich klar werden, was ps2 genau ist. Wir wissen, daß es ein Zeiger ist, und wir denken, daß wir seinen Namen kennen, aber der Name ist nicht ps2. Wäre dies der Fall, könnten Sie auch ps2 = 0; schreiben und das ist sicher

unmöglich (übersetzen Sie es, um es zu testen), da es keinen Bezeichner gibt, der nur ps2 heißt. Es gibt zwar s3.ps2 und einen Bezeichner ps3-> ps2, der über einen Zeiger zugreifen kann. Wir müssen also einen der beiden verwenden, wenn wir auf ps2 zugreifen wollen.

In Anlehnung an die (*Zeiger).Element Notation, ist die richtige Syntax für Zeile 33 in Zeile 37 zu sehen, da ps3->ps2 die richtige Zeigerreferenz auf ps2 ist. Das bedeutet nicht, daß Zeile 38 gleichbedeutend ist mit Zeile 37. Wie wir von vorhin wissen, erzeugt diese Zeile einen Fehler, wegen des Vorrangs des Operators *.

Abschließend sollten Sie sich klar machen, daß die Zeilen 32 und 37 nicht dieselbe Variable s1var anspricht, als Zeile 38. Ein s3tag enthält einen Zeiger auf ein s2tag und ein Vorkommen eines s2tag – dies ist nicht dasselbe. Sie können den Zeiger s2tag auf das Vorkommen von s2tag (s2inst) richten. Beabsichtigen Sie dies, so brauchen Sie die Zeilen 39 bis 42 nicht zu kodieren.

Strukturzugriff über mehrere Ebenen mit Funktionen

Es gibt viele Situationen, in denen Sie Funktionen aufrufen, die Strukturen oder Zeiger auf Strukturen zurückgeben. In diesen Fällen werden temporäre Variablen überflüssig. Sehen Sie sich die Funktion example1 in *Bild 11* an. Verstehen Sie, wieso alle Zeilen von 27 bis 30 12345 ausgeben? Zeile 27 sollte klar sein. Es handelt sich um eine Referenz s1.s1var, die statisch mit dem Wert 12345 initialisiert wurde. Zeile 28 sieht aus wie ein gewöhnlicher Strukturzugriff, mit demselben Vorrang und derselben Assoziativität der Operatoren. Sie kümmern sich nicht um die Klammern ()? Nein, da Sie die oberste Reihe der Operatorenhierarchie kennen. Da s1returner den Typ s1tag und explizit s1 zurückgibt, wieso sollte dies nicht identisch der Zeile 27 sein? Es handelt sich nur um einen Struktur.Element-Zugriff.

Zeile 29 zeigt eine Funktion, die einen Zeiger auf s1tag liefert. Es gibt aber auch hier keinen Unterschied, obwohl ein Funktionsaufruf beteiligt ist. Liefert die Funktion einen Zeiger, so greifen Sie wie bei Zeiger->Element zu. Natürlich ist Zeile 30 auch ein vertrauter Zeiger->Element-Zugriff, aber in der (*Zeiger).Element-Darstellung.

Diese Variablenzugriffe sind alle normal. Die Alternative wäre eine Codierung wie dies bei der Funktion example2 zu sehen ist. Es verkompliziert den Sachverhalt nur. Besonders fällt dies auf, wenn Sie eine zweite Struktur benötigen, wie bei s2 und im vorhergehenden Beispiel. So entstehen Zeiger und Referenzen zu allen möglichen Variablen. Dies artet in Programmen aus, wie Sie es in example3 und example4 (oder noch komplizierteren Situationen) sehen.

Strukturen und Malloc

Es gibt noch einen interessanten Punkt zu beachten. Normalerweise müssen Pakete von Informationen behandelt werden. Diese gliedern sich in Status- oder Kontrollinformationen, denen die eigentlichen Daten folgen. In der Regel tritt dies bei Kommunikationsprogrammen auf, muß aber nicht darauf beschränkt sein. Das Problem für den C-Programmierer besteht darin, daß der informelle Teil des Paketes fest und vorhersagbar ist. Der Datenanteil kann aber variable Länge besitzen. Es kann zum Beispiel der folgende Strukturteil die Kontrollinformationen beschreiben:

```
struct apacket {
       int    packethead;
       int    packetcontrol1;
       int    packetcontrol2;
       char   data[???];
};
```

Wie groß müssen wir aber die Dimension von data[] machen?

Machen wir es zu klein, so können wir Daten verlieren. Ist der Bereich zu groß, so werden unter Umständen wertvolle Ressourcen verschwendet. Wenn wir ihn vernünftig dimensionieren, so können wir nicht sicher sein, daß dies auch in der Zukunft richtig ist. Was ist also die beste Lösung? Wir können ein anderes Feld anfügen, das die Größe der Daten angibt, einige Unions deklarieren und ein Programm ähnlich dem Beispiel coderecord im vorherigen Artikel schreiben. Ich bin mir sicher, daß alle zustimmen werden, aber dies ist ein großes Unternehmen.

Die beste Lösung ist, dies alles zu umgehen, und mit dem zu arbeiten, was zur Verfügung steht. Da wir wissen, welche Elemente der Struktur die Kontrollinformationen beherbergen, warum sollen wir dies nicht ausnutzen? Dies und die Möglichkeit Speicher dynamisch durch die Standardbibliotheksfunktionen alloc, calloc und malloc zu allokieren, ist alles was wir benötigen.

Wir werden uns zuerst folgende Datenstruktur und folgendes Programmfragment überlegen:

```
struct apacket {
      int    packethead;
      int    packetcontrol1;
      int    packetcontrol2;
      int    packetsize;
      char    data[0];
};

<andere Programmteile>
struct apacket    apacket;
struct apacket    *ppacket;
ppacket = (struct apacket *)
malloc(sizeof(struct apacket)
+ apacket.packetsize);
```

Unglücklicherweise erlaubt C keine Datenelemente der Länge 0, auch nicht in Strukturkennzeichen. (Einige Compiler gestatten dies, was aber eindeutig ein Fehler ist. Es ist kein portables Konstrukt und sollte daher vermieden werden).

Viele von Ihnen werden sagen, daß ein eindimensionales Array mit einer Länge von 1 (zum

▶ *Bild 12: Zugriff über mehrere Ebenen mit Arrays.*

```
1  #define HBOUND(array)  (sizeof(array) /
2    sizeof(array[0]))
3
4  struct sltag {
5      char    charray[20];
6  };
7
8  struct sltag s1[10];
9  struct sltag *ps1;
10
11 main()
12 {
13     int     i;
14     int     j;
15     char    *cp;
16
17     printf("%d/%d=%d\n", sizeof(s1),
18                 sizeof(s1[0]), HBOUND(s1));
19
20     for (i = 0; i < HBOUND(s1); i++)
21    for (j = 0; j < sizeof(s1[0].charray); j++)
22        s1[i].charray[j] = '\0';
23
24     ps1 = &s1[5];
25     ps1->charray[2] = 5; /* Note that this is ASCII 5 */
26     (*ps1).charray[2] = 5; /* not '5'                 */
27
28     printf("%d\n", sizeof(ps1->charray));
29     for (ps1 = &s1[0]; ps1 < &s1[HBOUND(s1)];
30             ps1++) {
31    cp = ps1->charray;
32    while (cp < &ps1->charray[sizeof(
33                                 ps1->charray)])
34        *cp++ = '\0';
35     }
36 }
```

Beispiel char data[1]) ohne Probleme arbeitet (was der Fall ist). Der 1 Byte große Wert muß von der Größenangabe, die malloc übergeben wird, abgezogen werden. Diese Länge kann mit sizeof(char), oder sizeof(char[1]) angegeben werden, oder es braucht nur die Konstante 1 verwendet werden, da in diesem Fall dasselbe repräsentiert wird. Im zweiten Falle, da sizeof einen abgeleiteten Typen als Argument akzeptiert, sagt die Anweisung: Gib mir die Größe eines Arrays der Größe x Zeichen, wobei x 1 ist.

Ähnliches läßt sich durch das Macro offsetof erreichen. In diesem Falle enthält die Struktur auch das Element char data[1], aber anstatt:

```
struct apacket    apacket;
struct apacket    *ppacket;
ppacket = (struct apacket *)
malloc(sizeof(struct apacket)
+ apacket.packetsize);
```

schreiben wir:

```
ppacket = (struct apacket *)
malloc(offsetof(apacket.data)
+ apacket.packetsize);
```

da der Offset von data innerhalb von apacket der Länge der vorhergehenden Felder in apacket entspricht (in unserem Falle allen Feldern). Ich glaube, dies ist eine akzeptable Methode. Mit offsetof ist das Konstrukt verständlich. Nehmen wir die Größe der Struktur und addieren wir die Länge der gewünschten Daten. Nachdem wir den Zeiger von der gewünschten Länge haben, können wir die Strukturmitglieder kopieren. Bevor wir dieses Thema aber beenden, lassen Sie mich noch einmal auf den vorhergegangenen Artikel verweisen und Sie auffordern, die Stelle mit den Löchern und dem Packen von Strukturen nachzulesen. Wir müssen auch die Struktur, von der wir kopieren, so deklarieren.

Diese Art des Erzeugens dynamischer Strukturen ist, wie viele Dinge, nicht frei von Problemen und verlangt einige Vorüberlegungen. Hier müssen z.B. alle Referenzen auf die Elemente des Pakets über Zeiger durchgeführt werden.

Dies gilt auch für den variabel langen Datenbereich, der als einzelne Einheit übrigbleibt. Wenn Sie keine variabel lange Datenstruktur benötigen, so sollten Sie die Datenstruktur einfacher aufbauen. Sie können zum Beispiel die Struktur apacket so aufbauen, daß data kein Array, sondern ein Zeiger auf die Daten ist.

Diese Art ist natürlicher, da apacket-Strukturen initialisiert werden müssen, und dann referenziert werden können (char *data;). Auf alle übrigen Strukturelemente kann dann direkt mit dem Punkt-Operator zugegriffen werden. Die enge Verwandtschaft zwischen Arrays und Zeigern gestattet das Referenzieren in einer Array-Notation:

```
struct apacket somepaket;
somepaket.data = (char *)
malloc(somepaket.packetsize);
<...>
somepaket.data[i] = <...>;
<...>
*somepaket.data = <...>;
<...>
```

Der einzige Nachteil besteht darin, daß Sie den Speicherbereich, auf den somepacket.data zeigt, freigeben müssen (für dynamisch allokierten Speicher kontrolliert der Programmierer die Lebensdauer des Speicherbereichs).

Arrays

Da wir uns gerade mit dynamisch allokiertem Speicher beschäftigen, sollten wir uns noch ein bißchen über Arrays und Strukturen unterhalten. Sehen Sie sich bitte zuerst Zeile 21 von *Bild 12* an. Ich glaube, daß Sie keine Probleme haben, dies zu verstehen. Auch sollte es keine Schwierigkeiten geben, die Zeilen 24 bis 26 zu interpretieren, wo ein Zeiger auf eine Struktur auf ein Arrayelement gesetzt wird (der dann auf eine Struktur zeigt).

Arrays (und Zeiger) bedeuten, daß ihr Gebrauch derselbe ist, unabhängig, ob sie innerhalb einer Struktur oder als Strukturen verwendet werden (Zum Beispiel ändert sich die Syntax und Semantik bei der Übergabe eines Arrays an eine Funktion nicht, wenn das Array aus Strukturen statt aus Basistypen besteht).

Ich möchte Sie auch auf die Verwendung von HBOUND in Zeile 20 aufmerksam machen. Es ist einfach, wie Sie in den Zeilen 17 und 18 sehen, und beim Umgang mit Arrays sehr hilfreich.

Wenn Sie auf s1 über einen Zeiger zugreifen wollen, sehen Sie in den Zeilen 29 bis 35 eine Alternative zu den Zeilen 20 bis 22. Die unteren Zeilen sind sicherlich etwas schwerer verständlich, sie vermeiden aber die Indexnotation, was

vorteilhaft ist, wenn Sie auf andere Elemente der Struktur, oder auf dasselbe Element mehrmals zugreifen. Bezüglich HBOUND sollten Sie auch die Zeilen 29, 32 und 33 ansehen. Das Programm prüft, ob die Arraygrenzen verletzt werden, indem die Grenze mit dem Wert hinter dem letzten Eintrag verglichen wird. Ps1 wird also mit &s1[10] verglichen, das um eins größer als 9 ist (in C ist das erste Array-Element das Element 0). Ähnlich wird cp mit &ps1->charray[20] verglichen. Beachten Sie, wie dies ohne die Verwendung von Konstanten in den beiden verschachtelten Schleifen durchgeführt wird. Sie sollten solche Konstrukte in Ihren Programmen verwenden, unabhängig, ob Sie mit Strukturen arbeiten oder nicht.

Zusammenfassung

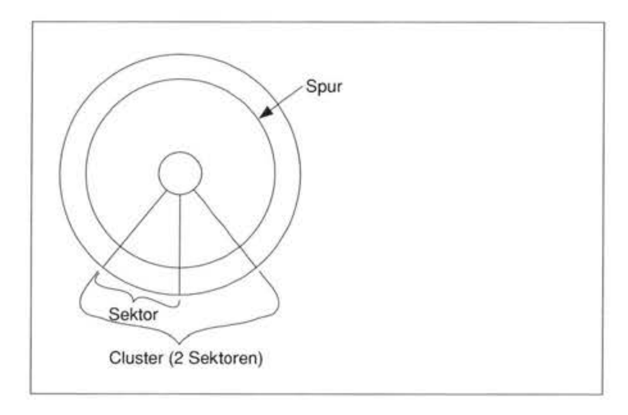
Obwohl dieser Artikel Strukturen, und verschiedene Aspekte von Zeigern auf Strukturen, Zugriffe auf Strukturelemente, Vermeidung von Speicherkonflikten bei Arrays und Speicherallokierung behandelt hat, trifft das Gesagte in großem Umfang auch auf Unions zu. Mit diesen Informationen, und denjenigen des vorausgegangenen Artikels, haben Sie eine fundierte Grundlage im Umgang mit Strukturen. Wenn Sie dies zu den Erkenntnissen über Unions, typedefs und C-Deklarationen hinzufügen, die Sie in den letzten Ausgaben erworben haben, können Sie hervorragend in C programmieren.

Greg Comeau

DOS-Know-how:

Disketten-struktur unter DOS

Jeder, der einen PC benutzt, arbeitet mit einer Festplatte und/oder Diskette. Unabhängig davon, welches System man besitzt, speichert man häufig Information auf Platten/Disketten. Da die Bedeutung von Festplatten/Disketten beim Betrieb eines Personalcomputers allgemein bekannt ist, würde man erwarten, daß die Benutzer wissen, wie Disketten funktionieren – aber das ist nicht der Fall! Wenn Sie vorhaben, auf den Disketten selbst zu arbeiten (auch wenn Sie nur ihre Struktur untersuchen wollen oder die Daten, die darauf gespeichert sind), dann sollten Sie wissen, wie sie arbeiten.



▸ *Bild 1: Eine Einzelplatte (zeigt eine Spur, einen Sektor und ein Cluster).*

Man kann sich eine Diskette als eine Ansammlung von Dateien vorstellen – nicht unähnlich einer Schublade in einem Dateikarten-Magazin. Jede Diskette enthält zahlreiche Dateien, und man kann auf jede Datei-»Karte« direkt zugreifen. In einer Dateikarten-Schublade findet man eine bestimmte Dateikarte, indem man den entsprechenden Reiter sucht, der ihren Inhalt angibt. Auf der Diskette ist die Dateistruktur einfach eine Interpretation der auf der Diskette gespeicherten Daten durch das Betriebssystem. Dateien existieren nicht auf Ebenen unterhalb von DOS.

Während des Formatierungsprozesses legt das Betriebssystem auf Ihrer Diskette die wohlbekannte Dateistruktur an. DOS erzeugt eine Index-Liste der Dateien (das Inhaltsverzeichnis, directory) und ein Verfahren, um einer Datei auf der Diskette Platz zuweisen zu können (die »file allocation«-Tabelle, FAT). DOS speichert die Information über den Aufbau der Diskette im Bootrecord, der auch auf Disketten ohne bootfähiges System vorhanden sein muß. Grundsätzlich besitzt jede Seite einer Platte/Diskette eine magnetisch beschichtete Oberfläche. Diese Oberfläche wird mittels eines »Lese-/Schreib«-Kopfes magnetisiert, der sich über die rotierende Diskette bewegt. Doppelseitige Disketten haben zwei speicherfähige Oberflächen; einseitige Disketten haben nur eine (obwohl beide Seiten beschichtet sind, erreicht nur eine den erforderlichen Qualitäts-Standard). Festplatten (hard disks) haben typischerweise zwei bis vier Platten, mit speicherfähigen Schichten auf jeder Seite.

Auf jedem Festplatten-/Diskettenlaufwerk wird der Lese-/Schreib-Kopf (auch mehrere) mit einem speziellen Motor über die Plattenoberfläche bewegt. Dieser Motor hat genau definierte Halt-Punkte (steps), an denen der Kopf zur Ruhe kommt. Jeder dieser Halt-Punkte definiert eine Spur, auf der Daten gespeichert werden können. Die meisten Festplatten bestehen aus einem Mehrfach-Plattensystem, auf welchem sich die Köpfe auf allen Platten zusammen bewegen. Die Spuren (auf allen Einzelplatten), die einem Schritt des Schritt-Motors entsprechen, werden Zylinder genannt.

Das Programm FORMAT unterteilt die Spuren in Sektoren von 512 Byte, um besser zu handhabende Plattensegmente zu erhalten: acht oder neun Sektoren pro Spur auf einer Diskette; siebzehn Sektoren pro Spur auf einer Festplatte.

DOS weist einer Datei Platz in Einheiten zu, die Cluster (Gruppierungen) genannt werden. Jedes Cluster besteht aus zwei bis acht Sektoren, abhängig vom Plattentyp. Wenn eine Datei zusätzlichen Plattenplatz braucht, so stellt das Betriebssystem dieser Datei ein oder mehrere zusätzliche Cluster zur Verfügung. *Bild 1* zeigt einen typischen Plattenaufbau.

Eine Platte/Diskette ist in die folgenden fünf wichtigen Abschnitte unterteilt:

```
ØØØØ  33 CØ 8E D8 8E CØ FA 8E-DØ BC ØØ 7C FB 8B F4 BF   3...........|....
ØØ1Ø  ØØ Ø6 B9 ØØ Ø1 FC F3 A5-5Ø B8 1E Ø6 5Ø CB B9 Ø4   ........P...P...
ØØ2Ø  ØØ BE BE Ø7 8Ø 3C 8Ø 74-2C 83 C6 1Ø E2 F6 B4 ØF   .....<.t,.......
ØØ3Ø  CD 1Ø B3 Ø7 BE 97 Ø6 B9-78 ØØ AC B4 ØE CD 1Ø E2   ........x.......
ØØ4Ø  F9 C7 Ø6 72 Ø4 34 12 B8-FF FF 5Ø 1E B4 ØF CD 1Ø   ...r.4....P.....
ØØ5Ø  32 E4 CD 16 CB B2 8Ø 8B-DC 8A 74 Ø1 BD Ø5 ØØ 8B   2.........t.....
ØØ6Ø  4C Ø2 B8 Ø1 Ø2 CD 13 73-1E 8Ø FC 11 75 1Ø 1E B8   L......s....u...
ØØ7Ø  ØØ FØ 8E D8 33 FF 81 7D-EA 43 4F 1F 74 Ø9 32 E4   ....3..}.CO.t.2.
ØØ8Ø  CD 13 4D 75 DD EB A7 81-BF FE Ø1 55 AA 75 9F 8B   ..Mu.......U.u..
ØØ9Ø  EE 1E 53 CB 43 52 4A ØD-ØA 45 72 72 6F 72 2Ø 6C   ..S.CRJ..Error l
ØØAØ  6F 61 64 69 6E 67 2Ø 6F-7Ø 65 72 61 74 69 6E 67   oading operating
ØØBØ  2Ø 73 79 73 74 65 6D 2Ø-66 72 6F 6D 2Ø 66 69 78    system from fix
ØØCØ  65 64 2Ø 64 69 73 6B 2E-ØD ØA ØD ØA 49 6E 73 65   ed disk.....Inse
ØØDØ  72 74 2Ø 43 4F 4D 5Ø 41-51 2Ø 44 4F 53 2Ø 64 69   rt COMPAQ DOS di
ØØEØ  73 6B 65 74 74 65 2Ø 69-6E 2Ø 64 72 69 76 65 2Ø   skette in drive 
ØØFØ  41 2E ØD ØA 5Ø 72 65 73-73 2Ø 61 6E 79 2Ø 6B 65   A...Press any ke
Ø1ØØ  79 2Ø 77 68 65 6E 2Ø 72-65 61 64 79 3A 2Ø Ø7 ØØ   y when ready: ..
Ø11Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø12Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø13Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø14Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø15Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø16Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø17Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø18Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø19Ø  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø1AØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø1BØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ 8Ø Ø1   ................
Ø1CØ  Ø1 ØØ Ø4 Ø4 51 E9 11 ØØ-ØØ ØØ A1 A2 ØØ ØØ ØØ ØØ   ....Q...........
Ø1DØ  41 EA Ø5 Ø4 D1 D2 B2 A2-ØØ ØØ 5D A2 ØØ ØØ ØØ ØØ   A.........]....
Ø1EØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø1FØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ 55 AA   ..............U.
                         Partition table
```

◂◂ Bild 2: Haupt-Bootrecord der Festplatte, Platten-Partitionstabelle.

- die Partitionstabelle
- den Bootrecord
- die Dateibelegungstabelle (file allocation table, FAT)
- das Inhaltsverzeichnis (directory)
- den Datenraum.

Sie werden in den folgenden Abschnitten besprochen.

Die Partitionstabelle

Jede Festplatte hat einen Haupt-Bootrecord (master boot record), der sich in Zylinder (Spur) 0, Kopf (Seite) 0, Sektor 1 befindet. Dieser Bootrecord ist zuständig für das Lesen und Entschlüsseln der Platten-Partitionstabelle, die am Ende des Haupt-Bootrecords enthalten ist. In Abhängigkeit vom Inhalt der Partitionstabelle ist der Haupt-Bootrecord dann zuständig für die Übergabe der Kontrolle an den Bootrecord des gerade bootfähigen Festplattenbereichs (hard disk partition).

Die Partitionstabelle beschreibt, wie die Festplatte unterteilt ist. Um von Programmen wie FDISK erkannt zu werden, muß die Partitionstabelle einem Standardaufbau genügen. Auf einer Festplatte können bis zu vier Aufteilungen existieren. Jede hat einen korrespondierenden Eintrag in der Partitionstabelle. *Bild 2* zeigt einen Speicherauszug (dump) des Haupt-Bootrecords eines COMPAQ Deskpro 286(RXXX). Beachten Sie die Partitionstabellen-Information, die am Ende des Sektors gespeichert ist. Die Einträge beginnen bei Offset 01BEh für Bereich 1, 01CEh für Bereich 2, 01DEh für Bereich 3 und 01EEh für Bereich 4. Die letzten beiden Byte des Sektors (welche unmittelbar auf die Partitionstabelle folgen, bei Offset 01FEh) sind ein Signatur-Wort für den Sektor, in diesem Fall AA55h.

Byte-Offset	Feld-Länge	Beispielwert	Bedeutung
00h	Byte	80h	Bootanzeige
			0h = nicht bootfähig
			80h = bootfähig
01h	Byte	01h	Startkopf
02h	Byte	01h	Startsektor
03h	Byte	00h	Startzylinder
04h	Byte	04h	System-ID
			00h=unbekannt
			01h=DOS,12-Bit-FAT
			04h=DOS,16-Bit-FAT
			05h=DOS,erweiterte Platte (extended disk), 16-Bit-FAT
05h	Byte	04h	Endkopf;
06h	Byte	51h	Endsektor;
07h	Byte	E9h	Endzylinder;
08h	DWort	0:0011	Erster Partition-Sektor
0Ch	DWort	0:A2A1	Sektoren in der Partition

◂ Tabelle 1: Layout eines Eintrags in die Partitionstabelle einer Festplatte.

Beachten Sie, daß die Partitionstabellen-Information in *Bild 2* nur zwei Einträge hat – es gibt nur zwei Bereiche auf dieser Festplatte. Jeder Eintrag in der Partitionstabelle ist 16 Byte lang. *Tabelle 1* zeigt den Aufbau jedes Eintrags in die Partitionstabelle, wobei die jeweiligen Werte der Partitionstabelle aus Bild 2 entnommen wurden.

Beachten Sie die in der Partitionstabelle gespeicherte Information. Der größte Teil dieser Daten besteht aus Grenzwerten für die jeweilige Partition. Zwei Felder, die Bootanzeige (boot indicator) und die Systemkennung (system ID), sind von besonderem Interesse. Die Bootanzeige signalisiert, ob diese Partition bootfähig ist. Nur eine der vier möglichen Partitionen kann als bootfähig gekennzeichnet werden. Die Systemkennung kennzeichnet den Partitionstyp (das Betriebssystem in der Partition). In *Tabelle 1* sind mehrere mögliche Werte für die Systemkennung aufgelistet. Durch verschiedene andere Betriebssysteme (XENIX, UNIX, Pick etc.) wird die Liste der möglichen Systemkennungen aber erweitert werden müssen.

Während der Systeminitialisierung (system boot) konsultiert das BIOS den ersten Sektor auf der Platte, um mit dem Booting-Prozeß fortzufahren. Bei der Diskette ist dies der Bootsektor (siehe den nächsten Abschnitt), bei der Festplatte der Haupt-Bootrecord, der bereits beschrieben wurde. Innerhalb dieses Haupt-Bootrecords befindet sich die Aufteilungstabelle (partition table), und das BIOS bestimmt (durch Setzen der Bootanzeigefelder), welche Partition bootfähig ist. Nach dem Auffinden der bootfähigen Partition wird die Kontrolle an den Bootsektor dieser Partition übergeben, und das Booting wird fortgesetzt wie bei einer Diskette.

Die Platten-Partitionierung erzeugt eine Anzahl von logischen Platten auf der Festplatte. Jede logische Platte verhält sich wie ein kleineres Plattenlaufwerk (dem vom Plattentreiber ein Laufwerkbuchstabe zugewiesen wird).

▶ *Bild 3: Der Aufbau des Bootsektors.*

▶▶ *Tabelle 2: Aufbau des BIOS-Parameter-Blocks (BPB).*

```
Jump instruction              System name            BPB

ØØØØ  E9 8D ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ Ø2 Ø2 Ø1 ØØ   ................
ØØ1Ø  Ø2 7Ø ØØ DØ Ø2 FD Ø2 ØØ-Ø9 ØØ Ø2 ØØ ØØ ØØ ØØ ØØ   .p..............
ØØ2Ø  ØØ DF Ø2 25 Ø2 ØF 2A FF-5Ø F6 ØØ Ø2 ØD ØA 4E 6F   ...%..*.P.....No
ØØ3Ø  6E 2D 53 79 73 74 65 6D-2Ø 64 69 73 6B 2Ø 6F 72   n-System disk or
ØØ4Ø  2Ø 64 69 73 6B 2Ø 65 72-72 6F 72 2E ØD ØA 52 65    disk error...Re
ØØ5Ø  7Ø 6C 61 63 65 2Ø 61 6E-64 2Ø 7Ø 72 65 73 73 2Ø   place and press 
ØØ6Ø  61 6E 79 2Ø 6B 65 79 2Ø-77 68 65 6E 2Ø 72 65 61   any key when rea
ØØ7Ø  64 79 ØD ØA Ø7 ØØ 49 4F-2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 53 59   dy....IO      SY
ØØ8Ø  53 4D 53 44 4F 53 2Ø 2Ø-2Ø 53 59 53 ØØ ØØ ØØ ØØ   SMSDOS   SYS....
ØØ9Ø  FC 33 CØ 8E D8 8E CØ FA-8E DØ BC ØØ 7C FB A1 78   .3..........|..x
ØØAØ  ØØ A3 8C 7C A1 7A ØØ A3-8E 7C 8D Ø6 21 7C A3 78   ...|.z...|..!|.x
ØØBØ  ØØ 8C 1E 7A ØØ E8 38 ØØ-A1 8E 7C A3 7A ØØ A1 8C   ...z..8...|.z...
ØØCØ  7C A3 78 ØØ 8A 16 2Ø 7C-EA ØØ ØØ 7Ø ØØ A1 8E 7C   |.x... |...p...|
ØØDØ  A3 7A ØØ A1 8C 7C A3 78-ØØ BE 2C 7C AC ØA CØ 74   .z...|.x..,|...t
ØØEØ  Ø9 B4 ØE BB Ø7 ØØ CD 1Ø-EB F2 B4 ØØ CD 16 CD 19   ................
ØØFØ  33 CØ CD 13 72 D7 8B 36-ØE 7C AØ 1Ø 7C B4 ØØ F7   3...r..6.|..|...
Ø1ØØ  26 16 7C Ø3 FØ 8B EE BB-2Ø ØØ A1 11 7C F7 E3 B1   &.|.........|...
Ø11Ø  Ø9 Ø5 FF Ø1 D3 E8 Ø3 E8-8D 1E ØØ Ø5 B8 Ø1 ØØ E8   ................
Ø12Ø  3B ØØ BE 76 7C BF ØØ Ø5-B9 ØB ØØ F3 A6 74 Ø2 EB   ;..v|........t..
Ø13Ø  9C BF 2Ø Ø5 B9 ØB ØØ F3-A6 74 Ø2 EB 9Ø BB ØØ Ø7   .. ......t......
Ø14Ø  8B F5 A1 1C Ø5 B1 Ø9 Ø5-FF Ø1 D3 E8 E8 ØE ØØ 8B   ................
Ø15Ø  1E 1C 7C Ø3 DD 8B ØE 1E-7C 83 D1 ØØ C3 96 Ø3 Ø6   ..|.....|.......
Ø16Ø  1C 7C 8B 16 1E 7C 83 D2-ØØ F7 36 18 7C 8B CA 41   .|...|....6.|..A
Ø17Ø  BA ØØ ØØ F7 36 1A 7C 8A-E8 BØ ØØ D1 E8 D1 E8 8A   ....6.|.........
Ø18Ø  E1 ØA C8 8A F2 8A 16 2Ø-7C AØ 18 7C 2A C4 FE CØ   ....... |..|*...
Ø19Ø  B4 ØØ 3B FØ 73 Ø2 8B C6-BF Ø3 ØØ 5Ø B4 Ø2 CD 13   ..;.s......P....
Ø1AØ  58 73 Ø6 4F 75 F5 E9 24-FF 2B FØ 74 26 8A EØ BØ   Xs.Ou..$.+.t&...
Ø1BØ  ØØ D1 EØ Ø3 D8 8Ø E1 CØ-8Ø C9 Ø1 8Ø C6 Ø1 3A 36   ..............:6
Ø1CØ  1A 7C 75 ØA B6 ØØ 8Ø C5-Ø1 73 Ø3 8Ø C1 4Ø A1 18   .|u......s...@..
Ø1DØ  7C EB BF C3 ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   |...............
Ø1EØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ   ................
Ø1FØ  ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ-ØØ ØØ ØØ ØØ ØØ ØØ 55 AA   ..............U.

                          Loader routine
```

Eine einzelne Festplatte kann nur ein einziges Betriebssystem, oder jedes logische Laufwerk ein anderes Betriebssystem enthalten. Computer haben oft MS-DOS in einer Partition und XENIX in einer anderen – man hat gewissermaßen zwei Computer zum Preis von einem.

Partitionierung ist oftmals nötig, wenn man Festplatten mit einer größeren Kapazität als 32 Mbyte verwendet. Einige Versionen von DOS sind auf 32 Mbyte oder weniger in einer einzelnen Partition begrenzt. Durch Benutzung mehrerer Partitionen kann man Platten bis zu 128 Mbyte verwenden. Kommerzielle Dienstprogramme stehen zur Verfügung, um die Beschränkung auf 32 Mbyte mit speziellen Treibern zu umgehen. Aber da die Platten vielfach nicht ohne die Treiber verwendet werden können, kann es zu Problemen führen, wenn man versucht, andere Betriebssysteme laufen zu lassen, oder von Disketten zu booten.

Der Bootrecord

Nachdem das System festgelegt hat, wohin der Bootrecord für den bootfähigen Plattenbereich geschrieben werden soll, lädt das BIOS den Bootrecord in den Speicher. Ein typischer Bootsektor für eine Diskette ist in *Bild 3* zu sehen.

Der Bootsektor beginnt mit einem Sprung zum Start der Bootstrap-Lader-Routine, welche mit »bootstrap« das System zum Laufen bringt. Das kleine Bootstrap-Programm wird geladen, und lädt wiederum selbst das größere Betriebssystem.

Auf den 3 Byte langen Sprungbefehl folgt ein 8 Byte langes Systemnamenfeld, das ausgefüllt werden kann, um den Hersteller anzuzeigen, mit dessen System die Diskette formatiert wurde (manche Hersteller tragen keinen Namen ein).

Byte-Offset	Feld-Länge	Beispielwert	Bedeutung
00h	Wort	0200	Byte pro Sektor
02h	Byte	02	Sektoren pro Cluster
03h	Wort	0001	Zahl der reservierten Sektoren, die bei Sektor 0 beginnen
05h	Byte	02	Zahl der FATs
06h	Wort	0070	Max. Zahl Einträge im Stammverzeichnis
08h	Wort	02D0	Gesamtzahl der Sektoren
0Ah	Byte	FD	Medien-Beschreiber
0Bh	Wort	0002	Zahl der Sektoren pro FAT
0Dh	Wort	0009	Zahl der Sektoren pro Spur
0Fh	Wort	0002	Zahl der Köpfe
11h	DWort	0:0000	Zahl der versteckten Sektoren
15h	11 Byte	—	Reserviert

Dann folgt der BIOS-Parameter-Block (BPB), der die Information liefert, die in *Tabelle 2* aufgelistet ist. Die Werte des Beispiels in Tabelle 2 stammen aus dem Bootsektor von Bild 3. (Diese Werte gehören zu einer 360 Kbyte DSDD Diskette.)

Dieser BPB ist wichtig für den Ablauf des Bootstrap-Programms, weil es diese Parameter kennen muß, um IO.SYS und MSDOS.SYS finden zu können (das BIOS und den Kern des Betriebssystems). Er ist auch die Grundlage dafür, was DOS über die Bootfähigkeit einer Diskette weiß.

Die Dateibelegungstabelle (File Allocation Table, FAT)

Die Dateibelegungstabelle (FAT) ist der von DOS benutzte Platz, von dem aus der Zugriff zum Datenbereich der Platte verwaltet wird. DOS verwendet die FAT, um anzuzeigen, welche Teile der Platte zu einer Datei gehören. Ältere Betriebssysteme benutzten kein dynamisches Konzept des Dateimanagements, sondern weisen jeder Datei einen Plattenbereich fester Größe zu. Da Dateien prinzipiell beliebig groß sein können, besteht der Nachteil dieser Methode darin, daß eine Datei fester Größe zumeist eine Platzvergeudung bedeutet.

Die FAT folgt auf der Platte dem Bootrecord. Da der Bootrecord nur einen Sektor lang ist (Sektor 0), beginnt die FAT mit Sektor 1. Die Länge der FAT (in Sektoren) kann sowohl aus dem Bootrecord selbst ermittelt werden, als auch aus der Anzahl der FAT-Kopien. Wegen der Wichtigkeit der FAT unterhält DOS gewöhnlich zwei Kopien, die sich dauernd nebeneinander auf der Platte befinden. Wird die Original-FAT verändert, bringt DOS die zweite Kopie gewissenhaft auf den neuesten Stand. Es ist interessant, zu erwähnen, daß kein einziges DOS-Kommando die zweite FAT-Kopie verwendet. Falls die Original-FAT irgendwie beschädigt ist, so kann ein eigenes Dienstprogramm verwendet werden, um

```
2D14:0100  FD FF FF 03 40 00 05 60-00 07 80 00 09 A0 00 0B   ....@...........
2D14:0110  C0 00 0D E0 00 0F 00 01-11 20 01 13 40 01 15 60   ............@..`
2D14:0120  01 17 F0 FF 19 A0 01 1B-C0 01 1D E0 01 1F 00 02   ................
2D14:0130  21 20 02 23 40 02 25 60-02 27 80 02 29 A0 02 2B   ! .#@.%`.'..)..+
2D14:0140  C0 02 2D E0 02 2F 00 03-31 20 03 33 40 03 35 F0   ..-../..1 .3@.5.
2D14:0150  FF 37 80 03 39 A0 03 3B-C0 03 3D E0 03 3F 00 04   .7..9..;..=..?..
2D14:0160  41 20 04 43 40 04 45 60-04 47 80 04 49 A0 04 4B   A .C@.E`.G..I..K
2D14:0170  C0 04 4D E0 04 FF 0F 00-00 00 00 00 00 00 00 00   ..M.............
2D14:0180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:01D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:01F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0200  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0280  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:0290  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:02A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:02B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:02C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:02D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:02E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2D14:02F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

◂◂ *Bild 4: Beispiel einer 12-Bit-FAT.*

mit der zweiten FAT-Kopie Plattendateien wiederherzustellen.

Die FAT ist aus einer Reihe von Byte aufgebaut, die benötigt werden, um den Status eines jeden Clusters auf einem Plattenlaufwerk anzuzeigen. Es gibt Codes, die anzeigen, ob ein Cluster zur Verfügung steht, gerade in Gebrauch ist, ob es reserviert oder beschädigt ist. Das genaue Code-Schema, das die FAT verwendet, hängt von der Kapazität des Plattenlaufwerks ab. Da jeder Eintrag in der FAT in der Lage sein muß, eine Clusternummer der Platte darzustellen, muß die Länge jedes FAT-Eintrags gleich der Anzahl der Bits sein, die nötig sind, um die größtmögliche Clusternummer der Platte darzustellen. DOS-Versionen vor 2.0 waren begrenzt auf 12-Bit-Einträge in der FAT; die größte Platte, die dargestellt werden konnte, umfaßte 2^{12} XXX(4096) Cluster. Da einige FAT-Eintragswerte benutzt werden, um den Status eines Clusters anzuzeigen, vermindert sich die tatsächliche Cluster-Anzahl, die in einer 12-Bit-FAT dargestellt werden kann, um 16 auf 4080. Wenn jedes Cluster vier 512-Byte-Sektoren umfaßt, dann begrenzt die 12-Bit-FAT die Größe der Platte auf 8355840 Byte – knapp unter 8 Mbyte. Obwohl 8 Mbyte Speicherplatz riesig erschien, als der PC eingeführt wurde, ist es gemessen an heutigen Standards nicht viel. Auch wenn die Zahl der Sektoren pro Cluster auf 8 erhöht wird, so erhöht sich die Plattenlimitierung nur auf knapp 16 Mbyte – eine ernsthafte Einschränkung von DOS.

Dieses Hindernis wurde dadurch umgangen, daß es DOS ermöglicht wurde, eine FAT zu verstehen, die anders codiert ist. Wenn das Plattenlaufwerk groß genug ist, kann DOS ab der Version 2.0 einen 16-Bit-FAT-Eintrag verwenden. Dieses 16-Bit-FAT erlaubt die Behandlung von 65536 Clustern (in Wirklichkeit nur 65520, wenn man die Cluster-Statuswerte in Betracht

Kategorie	Code
Frei für Zuweisung	0
Teil einer Datei (Zeiger auf die nächsten Cluster)	2-FEF
Reserviert	FF0-FF6
Fehlerhaftes Cluster	FF7
Ende der Cluster-Kette	FF8-FFF

◂ *Tabelle 3: 12-Bit-FAT, Zuordnungs-Byte der Cluster.*

zieht), oder 268369920 Byte (knapp unter 256 Mbyte), bei Benutzung von acht 512-Byte-Sektoren pro Cluster.

Wenn eine Platte formatiert wird, stellt das Programm FORMAT fest, welches Codierschema benutzt werden soll. Ergibt die Größe der Platte, daß sie mit einer 12-Bit-FAT dargestellt werden kann, dann wird dieses Schema benutzt; im andern Fall eine 16-Bit-FAT. Wir wollen uns jetzt die beiden FAT-Typen genauer ansehen.

Die 12-Bit-FAT

Die 12-Bit-FAT ist um 25 Prozent kleiner als die 16-Bit-FAT. Diese Tatsache war wahrscheinlich ausschlaggebend für den Einsatz der 12-Bit-FAT. Zwei 12-Bit-Zahlen können in drei Byte untergebracht werden. *Bild 4* zeigt einen Beispielsektor aus einer 12-Bit-FAT.

Beachten Sie den Aufbau der Dateibelegungstabelle. In diesem Beispiel werden die ersten beiden FAT-Einträge (die ersten drei Byte) verwendet, um Systeminformationen darzustellen. Deshalb sind die Cluster 0 und 1 des Datenbereichs ohne Zugriffsmöglichkeit der FAT. Auf die folgenden eineinhalb Byte (12 Bits) (ein FAT-Eintrag für Cluster 2) folgt der Eintrag für Cluster 3, usw. Beachten Sie die drei Byte beim Offset 0103h, die die FAT-Einträge für Cluster 2 und 3 sind. Man kann 034000 unter Verwendung der folgenden Formeln (alle Werte sind hexadezimal) in zwei separate FAT-Einträge unterteilen:

```
Eintrag 1 = (Byte2 AND 0F) * 10000 + Byte1
Eintrag 2 = Byte3 * 10 + (Byte2 AND F0)/10
```

Deshalb ist der FAT-Eintrag für Cluster 2

```
FAT Cluster 2 = (40 AND 0F) * 10000 + 03
              = (0) * 10000 + 03
              = 03
```

und der FAT-Eintrag für Cluster 3 ist

```
FAT Cluster 3 = 00 * 10 + ((40 AND F0)/10)
              = 0 + (40 / 10)
              = 04
```

Jeder FAT-Eintrag zeigt auf das nächste von der Datei belegte Cluster. Deshalb bilden die FAT-Einträge eine Kette; wenn alle Glieder dieser Kette zusammengefügt werden, dann bezeichnet die Kette die Cluster, die von einer bestimmten Datei belegt werden.

Andere Werte für FAT-Einträge stellen jedoch keine Folgeclusternummer dar; diese Werte geben vielmehr den Status des Clusters an. *Tabelle 3* stellt die möglichen Codes für einen FAT-Eintrag zusammen.

► *Bild 5: Ein Beispiel einer 16-Bit-FAT.*

►► *Tabelle 4: 16-Bit-FAT, Zuordnungs-Byte der Cluster.*

```
2D14:0100  F8 FF FF FF 03 00 04 00-05 00 06 00 07 00 08 00  ................
2D14:0110  09 00 0A 00 0B 00 0C 00-FF FF 0E 00 0F 00 10 00  ................
2D14:0120  11 00 12 00 13 00 14 00-15 00 16 00 17 00 18 00  ................
2D14:0130  19 00 1A 00 1B 00 FF FF-78 0A FF FF 20 00 FF FF  ........x... ...
2D14:0140  FF FF FF FF FF FF 6D 04-26 00 FF FF 27 00 28 00  ......m.&...'.(.
2D14:0150  29 00 2A 00 2B 00 2C 00-2D 00 2F 00 FF FF 32 00  ).*.+.,.-./...2.
2D14:0160  FF FF 00 00 33 00 51 00-42 00 36 00 37 00 38 00  ....3.Q.B.6.7.8.
2D14:0170  FF FF 3A 00 3B 00 3C 00-3D 00 3E 00 3F 00 41 00  ..:.;.<.=.>.?.A.
2D14:0180  FF FF FF FF 43 00 44 00-45 00 48 00 FF FF FF FF  ....C.D.E.H.....
2D14:0190  49 00 4A 00 4B 00 4C 00-4F 00 FF FF FF FF 50 00  I.J.K.L.O.....P.
2D14:01A0  FF FF 52 00 53 00 54 00-55 00 57 00 FF FF 81 00  ..R.S.T.U.W.....
2D14:01B0  59 00 5A 00 5B 00 5C 00-61 00 72 00 FF FF 60 00  Y.Z.[.\.a.r...`.
2D14:01C0  FF FF 62 00 6B 00 FF FF-FF FF 66 00 FF FF FF FF  ..b.k.....f.....
2D14:01D0  FF FF 6A 00 FF FF 6C 00-6D 00 78 00 6F 00 FF FF  ..j...l.m.x.o...
2D14:01E0  71 00 FF FF 83 00 FF FF-FF FF 76 00 77 00 FF FF  q.........v.w...
2D14:01F0  7B 00 FF FF FF FF 7C 00-7D 00 7E 00 7F 00 80 00  {.....|.}.~.....
2D14:0200  FF FF 82 00 D0 00 84 00-85 00 86 00 87 00 88 00  ................
2D14:0210  8C 00 8A 00 8B 00 FF FF-8D 00 8E 00 FF FF 00 00  ................
2D14:0220  00 00 00 00 93 00 94 00-95 00 96 00 97 00 98 00  ................
2D14:0230  99 00 9A 00 9B 00 9C 00-9D 00 C3 00 FF FF A0 00  ................
2D14:0240  A1 00 A2 00 A3 00 A4 00-A5 00 A6 00 A7 00 A8 00  ................
2D14:0250  A9 00 AA 00 AB 00 AC 00-AD 00 AE 00 AF 00 B0 00  ................
2D14:0260  B1 00 B2 00 B3 00 B4 00-C2 00 B6 00 B7 00 B8 00  ................
2D14:0270  B9 00 BA 00 BB 00 BC 00-BD 00 BE 00 BF 00 C0 00  ................
2D14:0280  C1 00 FF FF FF FF C4 00-C5 00 C7 00 FF FF C8 00  ................
2D14:0290  CF 00 CA 00 CB 00 CC 00-CD 00 CE 00 FF FF FC 01  ................
2D14:02A0  D1 00 D2 00 D3 00 D9 00-D5 00 D6 00 D7 00 D8 00  ................
2D14:02B0  FF FF DA 00 DB 00 DC 00-DD 00 DE 00 DF 00 E0 00  ................
2D14:02C0  E1 00 E2 00 E3 00 E4 00-E5 00 E6 00 E7 00 E8 00  ................
2D14:02D0  E9 00 EA 00 EB 00 EC 00-ED 00 EE 00 EF 00 F0 00  ................
2D14:02E0  F1 00 F2 00 F3 00 F4 00-F5 00 F6 00 F7 00 F8 00  ................
2D14:02F0  F9 00 FA 00 FB 00 FC 00-FD 00 FE 00 FF 00 00 01  ................
```

Kategorie	Code
Frei für Zuweisung	0
Teil einer Datei (Zeiger auf die nächsten Cluster)	2-FEF
Reserviert	FF0-FF6
Fehlerhaftes Cluster	FF7
Ende der Cluster-Kette	FF8-FFF

Unter Verwendung der 12-Bit-FAT aus Bild 4 wollen wir einer Cluster-Kette folgen. Der Directory-Eintrag einer Datei zeigt auf das erste Cluster, das von der Datei belegt ist. In dieser Abbildung zeigt der Directory-Eintrag für IBMBIO.COM (22100 Byte lang) auf die Startclusternummer 2. Wenn Sie sich den Eintrag für Cluster 2 anschauen, sehen Sie einen Zeiger auf Cluster 3 – und Cluster 3 zeigt auf Cluster 4 (erinnern Sie sich, wir haben das soeben ausgerechnet). Cluster 4 zeigt dann auf 5, welches auf 6 zeigt, und so weiter bis Cluster 18h erreicht ist. Hier ist der FAT-Eintrag FFFh, was anzeigt, daß das Ende der Cluster-Kette erreicht ist.

Die 16-Bit-FAT

Seit der Freigabe der DOS-Version 2.0 werden größere Festplatten unterstützt und eine FAT, die 16 Bits (zwei Byte) pro Eintrag benutzt. *Bild 5* zeigt das Beispiel eines Sektors aus einer 16-Bit-FAT.

Die Umsetzung dieser Dateibelegungstabelle erfolgt sehr viel direkter als mit der 12-Bit-FAT. Die ersten beiden Einträge werden für Systeminformation genutzt; jeder benachbarte Eintrag belegt zwei Byte. Beachten Sie die zwei Byte beim Offset 0104h (den FAT-Eintrag für Cluster 2). Der Wert hier (0003h) zeigt auf den Eintrag für Cluster 3.

Jeder FAT-Eintrag zeigt auf das nächste Cluster, das von der Datei besetzt wird. Auf diese Weise bilden die FAT-Einträge eine Kette, welche, wenn alle Glieder zusammengefügt werden, die von einer bestimmten Datei besetzten Cluster anzeigt. Wie bei der 12-Bit-FAT stellen die übrigen FAT-Eintragswerte keine Folgeclusternummer dar, sondern zeigen den Status des Clusters an. *Tabelle 4* stellt die möglichen Codes für einen FAT-Eintrag zusammen.

Wir wollen eine Cluster-Kette durchlaufen, indem wir die 16-Bit-FAT verwenden, die in *Bild 5* dargestellt ist. Der Directory-Eintrag einer Datei zeigt auf das erste Cluster, das von der Datei besetzt ist. In dieser Abbildung zeigt der Directory-Eintrag für IBMBIO.COM (22100 Byte lang) auf eine Startclusternummer 2. Der Eintrag für Cluster 2 zeigt auf Cluster 3. Cluster 3 zeigt auf Cluster 4, das auf Cluster 5 zeigt, welches auf 6 zeigt, usw. bis Cluster 0Ch erreicht ist. Der FAT-Eintrag bei Cluster 0Ch ist FFFFh, was anzeigt, daß das Ende der Cluster-Kette erreicht wurde.

Weitere FAT-Informationen

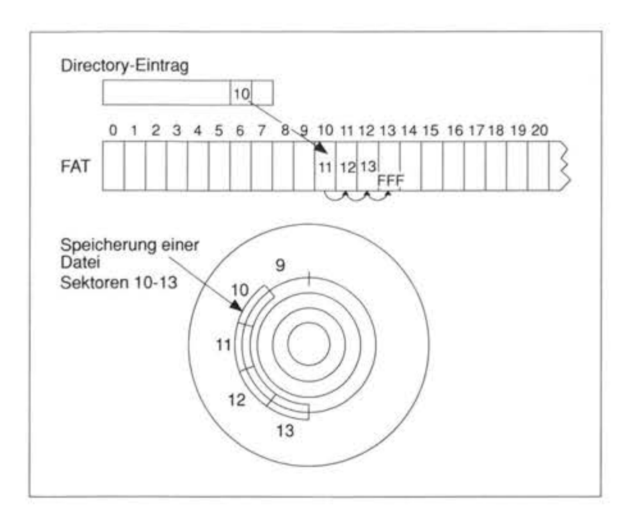
Wenn DOS Platz für eine Datei anfordert, so wird dieser Speicherplatz der Datei in Einheiten von einem oder mehreren Clustern zugewiesen. Wie Sie bei der Besprechung der 12- und 16-Bit-FAT gesehen haben, sind die Cluster in einer Datei verkettet, wobei jeder Eintrag in die FAT die Clusternummer des nächsten Eintrags angibt (siehe *Bild 6*).

Die FAT reserviert den Platz für die Einträge 0 und 1, ohne ihn zu nutzen. Das erste Byte der FAT wird für eine Diskettenkennung (ID, identification) verwendet, die dazu dient, das Diskettenformat festzustellen (siehe *Tabelle 5*). Da die Cluster 0 und 1 für das System reserviert sind, ist Cluster 2 das erste Cluster, das zugewiesen werden kann.

Beachten Sie, daß DOS den von Dateien benötigten Platz nur in Portionen ganzer Cluster zuteilt. Deshalb ist der kleinste von einer Datei nutzbare Plattenbereich, unabhängig von ihrer Größe, ein Cluster. Eine 1 Byte große Datei kann 512, 1024, 2048 oder 4096 Byte belegen, abhängig von der Anzahl Sektoren pro Cluster.

Wie Sie bei der Decodierung des BPB (BIOS-Parameter-Block, siehe Tabelle 2) einer Diskette gesehen haben, hatte die Diskette im Beispiel zwei FATs (Dateibelegungstabellen). Jedesmal, wenn Plattenoperationen Plattenplatz zuweisen oder freigeben, werden beide FATs automatisch aktualisiert. Wenn zum ersten Mal auf eine Platte zugegriffen wird, dann vergleicht DOS die FATs, um zu sehen, ob sie konsistent sind. Obwohl es mehr als zwei FATs geben kann – in diesem Fall werden sie sequentiell auf der Platte gespeichert – haben die meisten Platten zwei.

Nach den FATs kommt das Stammverzeichnis, mit 32 Byte für jeden Eintrag. Der BPB liefert die Größe des Inhaltsverzeichnisses, so daß man feststellen kann, wo der Dateibereich beginnt.



◂◂ *Bild 6: Cluster-Verkettung in der FAT.*

Jetzt, da Sie wissen, wo man etwas auf der Platte findet, wollen wir uns anschauen, welche Funktionen DOS zur Plattenverwaltung zur Verfügung stellt.

Der Gebrauch von Funktionen für Festplatte/Diskette

Es gibt keine BIOS-Funktionen für den Umgang mit einem DOS-Dateisystem, denn das Dateisystem (zusammen mit allen Tabellen, die gerade besprochen wurden) ist eine Konstruktion von DOS. Für das BIOS stellt die Platte nur eine Reihe von Sektoren dar, beginnend bei Sektornummer 0 und sequentiell fortlaufend bis zur höchsten Sektornummer. Das BIOS kennt sich aus mit Spuren, Sektoren und Plattenköpfen, aber nicht mit Dateien, FATs oder Inhaltsverzeichnissen. Alle Funktionen, die Sie anwenden werden, sind DOS-orientierte Funktionen.

Wert	Diskettencharakteristik
F0	Nicht identifizierbar
F8	Festplatte
F9	Doppelseitig, 15 Sektoren/Spur
F9	Doppelseitig, 9 Sekt./Spur (720 Kbyte)
FC	Einseitig, 9 Sektoren/Spur
FD	Doppelseitig, 9 Sekt./Spur (360 Kbyte)
FE	Einseitig, 8 Sektoren/Spur
FF	Doppelseitig, 8 Sektoren/Spur

◂◂ *Tabelle 5: Einige mögliche Werte für das ID-Byte der FAT.*

Laufwerksdaten

Sie können DOS-Funktionsaufrufe verwenden, um Information über das Platten-/Diskettenlaufwerk zu erhalten. Das Programm `drvinfo.c` zeigt, wie Sie diese Information erhalten und auf dem Bildschirm darstellen können (siehe dazu *Listing 1*).

Um Information über das Laufwerk zu erhalten, ruft `drvinfo.c` drei Subroutinen auf:

1. `get_drive()`, um die aktuelle Laufwerknummer zu erhalten
2. `get_drvinfo()`, um allgemeine Information über das Laufwerk zu erhalten
3. `get_drvspace()`, um andere Information zu erhalten

All diese Informationen werden folgendermaßen in die Struktur `drvinfo` eingetragen (in der Datei `drvinfo.h`):

```
struct drvinfo {
     int     spc;          /* Sektoren pro Cluster */
     int     avail;        /* Verfügbare Cluster */
     int     fatseg;       /* FAT-Segment des ID-Byte */
     int     faroff;       /* FAT-Offset des ID-Byte */
     int     secsize;      /* Physikalische Sektor-Größe */
     int     clusters;     /* Anzahl der Cluster */
     char    fatid;        /* ID-Byte der FAT */
} info;
```

Durch Definition einer Struktur für zusammenhängende Plattenninformation können Sie die Information logisch organisiert zusammenhalten, wenn Sie damit arbeiten.

Beachten Sie, daß das Programm `drvinfo.c` (siehe Listing 1) zwei Subroutinen-Aufrufe enthält, die zeigen, daß dieselbe Information auf verschiedene Weise erhalten werden kann.

◂ *Listing 1*

```
#include <stdio.h>
#include "drvinfo.h"

main(argc,argv)

int  argc;
char *argv[];

{
     int  drive;

 /*  Holt Daten ueber das aktuelle Laufwerk und gibt sie am
     Bildschirm aus */

     drive = get_drive();
     printf("Code des aktuellen Laufwerks = %d\n",drive);
     printf("Laufwerk ist %c:\n",'A'+drive);

     /*   Ermittelt folgende grundlegende Laufwerk-
          Information:
                Sektoren pro Cluster auf dem Laufwerk
                Physikalische Sektorgroesse
                Zahl der Cluster auf der Disk
          Mit dieser Information wird die Kapazitaet der
          Disk berechnet */

     get_drvinfo(*argv[1],&info);
     printf("Information fuer Laufwerk %c:\n",*argv[1]);
     printf("   Zahl der Sektoren pro Cluster =
                %d\n",info.spc);
     printf("   Groesse eines physikal. Sektors =
                %d\n",info.secsize);
     printf("   Zahl der Cluster = %d\n",info.clusters);
     printf("\n");
     printf("   Laufwerk-Kapazitaet ist = %dK\n",
          info.clusters * ((info.spc * info.secsize)/1024));
     printf("\n\n");

     /* Ermittelt folgende grundlegende Laufwerk-Information:
                Sektoren pro Cluster auf dem Laufwerk
                Physikalische Sektorgroesse
                Zahl der Cluster auf der Disk
          Dieser Aufruf holt zusaetzlich die Zahl der
          freien Cluster. Wir koennen jetzt sowohl die
          Kapazitaet als auch den freien Platz berechnen.
     */
     get_drvspace(*argv[1],&info);
     printf("Information fuer Laufwerk %c:\n",*argv[1]);
     printf("   Zahl der Sektoren pro Cluster = %d\n",info.spc);
     printf("   Groesse eines physikal. Sektors = %d\n",info.secsize);
     printf("   Zahl der Cluster = %d\n",info.clusters);
     printf("   Zahl der verfuegbaren Cluster = %d\n",info.avail);
     printf("\n");
     printf("   Laufwerk-Kapazitaet ist = %dK\n",
          info.clusters * ((info.spc * info.secsize)/1024));
     printf("   Verfuegbarer Platz ist = %dK\n",
          info.avail * ((info.spc * info.secsize)/1024));
}
```

► *Listing 2*

```
#include <stdio.h>
#include <dos.h>
#include "drvinfo.h"

get_drvinfo(drv,info)

char drv;
struct drvinfo *info;

{
     union REGS regs;
     struct SREGS segs;
     int dn;

     /* Konvertiert Laufwerk-Buchstabe in interne Darstellung */
     drv = toupper(drv);
     dn = drv - 'A' + 1;
     /* Initialisierung und Aufruf von DOS */
     regs.h.ah = 0x1c;
     regs.h.dl = dn;
     intdosx(&regs,&regs,&segs);
     info->spc = regs.h.al;
     info->fatseg = segs.ds;
     info->fatoff = regs.x.bx;
     info->secsize = regs.x.cx;
     info->clusters = regs.x.dx;
}
```

►► *Listing 3*

```
#include <stdio.h>
#include <dos.h>
#include "drvinfo.h"
get_drvspace(drv,info)
char drv;
struct drvinfo *info;
{
     union REGS regs;
     struct SREGS segs;
     int  dn;
     /* Konvertiert Laufwerk-Buchstabe in interne
        Darstellung */
     drv = toupper(drv);
     dn = drv - 'A' + 1;
     /* Initialisiere und mache den DOS-Aufruf */
     regs.h.ah = 0x36;
     regs.h.dl = dn;
     intdosx(&regs,&regs,&segs);
     info->spc = regs.x.ax;
     info->avail = regs.x.bx;
     info->secsize = regs.x.cx;
     info->clusters = regs.x.dx;
}
```

Um die Laufwerkinformation zu erhalten, wird ein simples Verfahren angewandt – man ruft den »zentralen« DOS-Interrupt (Int 21h) auf. Dieser Aufruf wird in C durch die Funktion `intdos` ausgeführt. Der DOS-Interrupt gibt den Laufwerkcode in Register AL zurück.

Die Funktion `get_drive()` interpretiert den Laufwerkcode nicht; sie gibt auf folgende Weise einfach den Code an `drvinfo.c` zurück:

```
#include <stdio.h>
#include <dos.h>
get_drive()
{
 union REGS regs;
 regs.h.ah = 0x19;
 intdos(&regs,&regs);
 return(regs.h.al);
}
```

Aber das Verfahren der Abfrage der Laufwerkinformation ist komplexer als eine simple Abfrage der Laufwerknummer. Die Information wird sowohl in den Segmentregistern zurückgegeben als auch in den allgemeinen Registern. Für den Zugriff auf die Segmentregister muß man die Funktion `intdosx` verwenden (siehe *Listing 2*).

Die Funktionen `get_drvinfo()` und `get_drvspace()` sind Beispiele eines guten Programmierstils, der besagt, daß Implementations-Details innerhalb der Funktionen versteckt werden sollen. In diesem Fall bestimmen die Funktionen den korrekten Laufwerkcode aus der benutzerorientierten Standardbezeichnung der Laufwerke (A:, B:, C: usw.). Indem erlaubt wird, den Laufwerknamen als einen Buchstaben zu übergeben, verbirgt man auf diese Weise die Tatsache, daß die Laufwerkbezeichnungen in DOS- und BIOS-Routinen nicht immer konsistent sind. Einige Routinen verwenden 0 zur Bezeichnung von Laufwerk A, während andere 0 zur Bezeichnung des Standardlaufwerks (default drive) verwenden.

Funktionen (wie `get_drvinfo()` und `get_drvspace()`), die in eine Bibliothek geschrieben werden, um vielen Programmierern zur Verfügung zu stehen, sollten soviele Implementations-Details verstecken, wie nur möglich. Diese Art »programmiererfreundlicher« Bibliotheksroutine hilft Programmierern, Programme zu erstellen, ohne sich um die Eigenheiten des Rechners kümmern zu müssen.

Die Funktion `get_drvinfo()` in Listing 2 ist gebrauchsfertig. Sie wurde zum Einbau in eine Funktionsbibliothek geschrieben und kann einzeln zu einem Objektmodul kompiliert werden.

Nachdem die Funktion (über den Funktionsaufruf `intdosx`) DOS aufgerufen hat, rettet sie die zurückerhaltene Information aus den Registern in die Struktur `info`, die mit dem Funktionsaufruf übergeben wurde. Indem die Information in diese Struktur gepackt wird, ermöglicht man jemandem ohne Kenntnisse über Register und DOS-Aufrufe, über das Programm, das die Funktion aufruft, darauf zuzugreifen. Die Funktion kann in eine Bibliothek eingebaut werden für Programmierer, die nicht wissen, wie man mit DOS umgeht.

Die DOS-Funktion 36h gestattet auf eine andere, vollständigere Art, Auskunft über die Platte zu erhalten. Die Funktion `get_drvspace()`, in *Listing 3*, lädt nicht nur Platteninformation in die Struktur `info`, sondern gibt auch den auf der Platte verfügbaren Platz zurück. Diese Funktion ist nützlich für Programme, die mit großen Plattendateien oder mit vielen Dateien arbeiten.

Um festzustellen, wieviel freier Platz auf der Platte zur Verfügung steht, benutzen Sie folgende Information (die man von der DOS-Funktion 36h erhält):

Reg.	**Inhalt**
AX	Anzahl der Sektoren pro Cluster
BX	Anzahl der verfügbaren Sektoren
CX	Byte pro Sektor
DX	Cluster pro Laufwerk

Zur Berechnung des freien Platzes auf dem Laufwerk verwenden Sie die folgende Formel:
BX * AX * CX

```
#include <stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
 int avail, total;
 get_free(*argv[1],&avail,&total);
 if (*argv[1])
  printf("Disk: Freier Platz auf Laufwerk %c: ist %dK von
          %dK\n",
   *argv[1],avail,total);
 else
  printf("Disk: Freier Platz auf Stand.-Laufw. ist %dK von
          %dK\n",
   avail,total);
}
```

Die gesamte Laufwerkkapazität wird nach folgender Formel berechnet:
DX * AX * CX

Wenn man nur herausfinden will, wieviel Platz auf der Platte frei ist, dann kann man eine Funktion (`get_free()`) schreiben, die nur diese Information liefert. Das Programm `free.c` erhält diese Information, indem es `get_free()` mit dem Laufwerknamen aufruft, und mit Zeigern auf Integers für den verfügbaren und den gesamten Plattenplatz.

Das Programm wurde so geschrieben, daß im ersten Kommandozeilen-Argument der Laufwerkname steht. Wenn kein erstes Argument eingegeben wird, dann nimmt das Programm an, daß es über das Standardlaufwerk (default drive) Auskunft geben soll.

Die Funktion `get_free()` stellt fest, welches Laufwerk überprüft werden soll, setzt dann entsprechend die Register und macht den Aufruf an DOS (siehe *Listing 4*). Die Funktion 36h wird wiederum verwendet, um den freien Platz festzustellen, aber die meiste Information über die Platte wird weggeworfen, denn sie wird für den eng umgrenzten Zweck der Funktion nicht gebraucht.

Wenn Sie ein wirklich nützliches Dienstprogramm für sich selbst schreiben wollen, warum dann nicht eines, das eine Gruppe von Dateien auf eine Diskette kopiert, oder auf eine Reihe von Disketten. Ein solches Dienstprogramm prüft die Größe der nächsten Datei, die Sie kopieren wollen. Wenn genügend Platz auf der Diskette zur Verfügung steht, kopiert das Dienstprogramm die Datei, wenn nicht, fordert es eine weitere Diskette an. Ein solches Programm könnte folgendermaßen aussehen:

```
FOR i=1 TO zahl_der_argumente
     TOP:
     groesse = Größe von Datei i
     platz = hole Platz auf Ziel-Laufwerk
     IF groesse > platz THEN
          Fordere eine weitere Diskette an
          Warte bis der Benutzer eine Taste gedrückt hat
          GOTO TOP:
     ELSE
          Kopiere Datei i auf Diskette
     ENDIF
NEXT i
```

Wenn Sie kommerzielle Anwendungen schreiben, dann ist Ihr Programm für das Vermeiden aller vorhersehbaren Fehler verantwortlich.

◄◄ *Programm: Free.c.*

◄ *Listing 4*

```
#include <stdio.h>
#include <dos.h>

get_free(drv,avail,total)

char drv;
int *avail,
    *total;

{
     union REGS regs;
     struct SREGS segs;
     int  dn;

  /* Bestimmt das Laufwerk und setzt die Laufwerk-Nummer */
     if(drv){
        drv = toupper(drv);
        dn = drv - 'A' + 1;
     } else {
        dn = 0;
     }
     /* Initialisiert und macht den DOS-Funktionsaufruf */
     regs.h.ah = 0x36;
     regs.h.dl = dn;
     intdosx(&regs,&regs,&segs);
     *avail = ((regs.x.ax * regs.x.cx)/1024) * regs.x.bx;
     *total = ((regs.x.ax * regs.x.cx)/1024) * regs.x.dx;
}
```

Immer wenn Sie Anwendungsprogramme schreiben, in denen eine Datei durch zusätzliche Daten vergrößert werden soll, laufen Sie Gefahr, auf der Platte dafür keinen Platz mehr frei zu haben. Überprüfen Sie unter allen Umständen den noch zur Verfügung stehenden Platz, um Plattenabstürze zu vermeiden!

Formatieren von Platten und Disketten

Die Formatierung von Platten/Disketten ist eine einfache aber gefährliche Aufgabe, die die meisten Leute nicht selbst programmieren müssen. Das Basisprogramm FORMAT, das mit DOS zusammen vertrieben wird, ist eines von mehreren adäquaten Plattenformatierungsprogrammen auf dem Markt. Spezielle Formatierer stehen zur Verfügung (oft als Teil eines Dienstprogramm-Pakets wie z.B. PCTOOLS) für diejenigen, die schnellere oder hochentwickeltere Formatierung wollen.

Anfänger sollten sorgfältig planen, bevor sie Plattenformatierungsprogramme schreiben, denn diese Programme können, wenn sie unsachgemäß gehandhabt werden, wichtige Plattensysteme löschen und monatelange Arbeit zerstören. In diesem Abschnitt werde ich einige grundlegende Formatierungstechniken beschreiben und mit einigen Beispielen zeigen, wie man die im System vorhandenen Formatierungsroutinen benutzt. Aber denken Sie daran: Sie bringen Ihr System in Gefahr, wenn Sie hier einen Fehler machen! Wenn Sie ein Formatierprogramm testen, befolgen Sie die folgenden einfachen Vorsichtsmaßnahmen:

• Wann immer es möglich ist, testen Sie ein Formatierungsprogramm auf einem »Nur Diskette«-System (ohne Festplatte). Verwenden Sie Systemdisketten, deren Verlust Sie sich leisten können, falls etwas schief geht.

• Wenn Sie Testläufe auf einem System ausführen müssen, das eine Festplatte enthält, schalten

► *Tabelle 6: Register-Belegung für BIOS Int 13h, Funktion 05h.*

►► *Bild 7: Plattenspur.*

Reg.	Bedeutung
AH	05h (der Funktionscode)
ES:BX	Zeiger auf die Tabelle des Spuren Adreßfelds
CH	Spurnummer
DH	Kopfnummer
DL	Laufwerknummer

Sie die Festplatte aus, wenn Sie können (indem Sie z.B. den Festplattencontroller herausziehen).

• Vergewissern Sie sich, daß Sie ein aktuelles Backup Ihres Betriebssystems haben (Sie sollten immer ein aktuelles Backup haben!)

Es gehört zu einem guten Programmierstil, mögliche Fehler einzukalkulieren und entsprechend »abzufangen«. Wenn Sie vorsichtig sind, kann Ihrem Betriebssystem, wenn Sie Ihre Programme testen, nichts passieren.

DOS stellt eine BIOS-Funktion (Int 13h, Funktion 05h) zur Formatierung von Diskettenspuren zur Verfügung. Das Konzept ist einfach, und die Funktion ist leicht zu handhaben. Wir wollen uns jetzt anschauen, wie man diese Funktion zum Formatieren einer Diskette benutzt.

Wenn Sie mit einer unformatierten Diskette anfangen (oder wenn Sie eine alte löschen wollen), müssen Sie Int 13h Funktion 05h verwenden, um die Diskette Spur für Spur folgendermaßen zu formatieren:

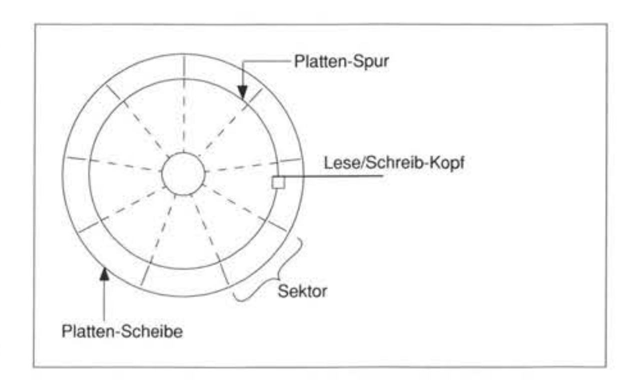
Für jede Spur von 0 bis zur letzten Spur
Richte den Aufruf für die Spur ein
Rufe die Routine zur Spurformatierung auf

Die Diskette wird korrekt formatiert, und BIOS-Routinen werden sie lesen können – aber es ist keine DOS-Diskette. Wie Sie sich vielleicht auf Grund der früheren Diskussion in diesem Artikel erinnern werden, verlangt DOS, daß eine Diskette bestimmte Strukturen besitzt (wie z.B. den Bootsektor, die FAT und das Stammverzeichnis). Die Formatprozedur stellt keine davon zur Verfügung.

Um eine für DOS akzeptable Diskette einzurichten, genügt es nicht, einer Diskette ihr charakteristisches Format zu geben, sondern man muß auch die Diskettenstruktur wie folgt initialisieren:

Für jede Spur von 0 bis zur letzten Spur
Setze die Aufrufparameter für die Spur-Formatierungsroutine
Rufe die Routine zur Spurformatierung auf
Schreibe den Bootsektor auf die Diskette
Schreibe die FAT-Information auf die Diskette
Schreibe die Stammverzeichnis-Information auf die Diskette

Man kann die letzten beiden Schritte dadurch erledigen, daß man einfach Nullen in die Bereiche von FAT und Diskettenverzeichnis schreibt. Einträge von Nullen in der FAT zeigen an, daß die Cluster frei sind und bereit für eine Neuzuweisung sind.



Nullen im Disketteninhaltsverzeichnis zeigen an, das die Verzeichniseinträge niemals benutzt wurden. Wenn man einmal vom Bootsektor absieht, dann kann der Formatierungsprozeß ziemlich einfach sein: Formatiere die Spuren, schreibe den Bootsektor, und anschließend setze die Bereiche von FAT und Stammverzeichnis auf Null.

Wir wollen jetzt eine Routine zum Formatieren einer Diskettenspur schreiben. Sie können dann eine Diskette formatieren, indem Sie einfach nacheinander durch alle Spuren »springen«.

Man ruft BIOS Int 13h, Funktion 05h mit der Register-Belegung auf, wie sie in *Tabelle 6* gezeigt wird.

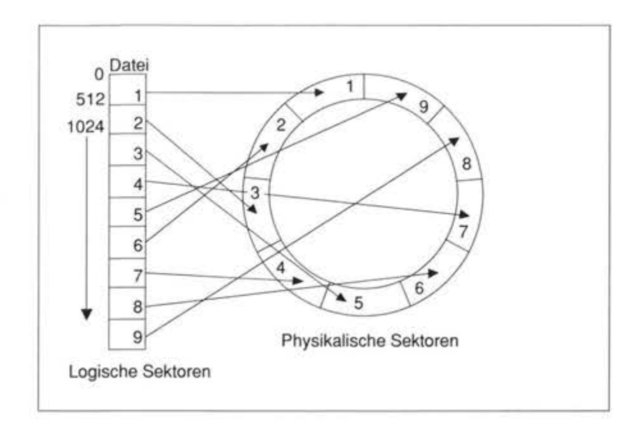
Die Spuradreßtabelle ist das Herz der Formatierungsoperation. Sie spezifiziert die Anordnung der logischen Diskettensektoren auf der physikalischen Diskettenspur. Jeder Diskettensektor wird in der Tabelle durch einen vier Byte langen Eintrag dargestellt, die Kennzeichnung jedes Sektors auf der Spur. Man kann die Tabelle zur Zuweisung logischer Sektornummern in einer anderen Anordnung als die der physikalischen Sektoren auf der Diskette verwenden (dieser Prozeß ist als »Verschränkung« oder »interleaving« bekannt).

Die Zahl der Spuren variiert entsprechend dem Diskettentyp, den man verwendet: 5 $^1/_4$-Zoll-Disketten (360 Kbyte, doppelseitig, doppelte Dichte) haben 40 Spuren pro Seite; 3 $^1/_2$-Zoll-Disketten (720 Kbyte, doppelseitig, doppelte Dichte) haben 80 Spuren pro Seite. Die Kopfnummer (für Disketten) sollte 0 oder 1 sein.

Die Laufwerknummer (Register DL) gibt an, mit welchem physikalischen Laufwerk man arbeiten will. (Physikalische Laufwerke werden ab Null durchnumeriert: Laufwerk A: entspricht Nummer 0, Laufwerk B: ist Nummer 1, usw.)

Auf einer unformatierten Platte/Diskette ist eine Spur ein unstrukturierter blanker Abschnitt der magnetischen Oberfläche. Die Formatierprozedur prägt der Platte/Diskette eine Struktur auf, indem magnetisch »Speicherbehälter« (storage bins) auf der Spur erzeugt werden (siehe *Bild 7*). In diesen Behältern, die Sektoren genannt werden, kann Information gespeichert werden.

Man kann die Sektoren, physikalisch aufeinanderfolgend, rund um die Spur plazieren, aber diese Methode hat Nachteile.



◄◄ *Bild 8: Speicherung von Dateisektoren.*

Code	Sektorgröße (in Byte)
0	128
1	256
2	512
3	1024

◄ *Tabelle 7: Größencodes.*

Stellen Sie sich vor, es ist eine Anzahl von Plattensektoren zu lesen. Sie wollen beispielsweise einen Plattensektor in den Arbeitsspeicher kopieren, kurz damit arbeiten, und dann zurückkommen, um den nächsten Sektor zu lesen. Es ist kein Problem: Sage dem Plattencontroller einfach, welchen Sektor du willst, und der Controller liefert den richtigen. Aber was geschieht, wenn die beiden Sektoren auf der Platte benachbart sind? Ihre Anforderung des zweiten Sektors wird erfolgen, nachdem der Anfang dieses Sektors den Lese-/Schreib-Kopf passiert hat. Um den Sektor zu erhalten, müssen Sie warten, bis die Platte eine ganze Umdrehung ausgeführt hat (ca. eine Fünftelsekunde für eine Diskette bei 300 Umdrehungen/Minute). Dieser Zeitbetrag summiert sich.

Wenn Sie versuchen, eine ganze Platte zu lesen, einen Sektor nach dem anderen, dann summieren sich diese Fünftelsekunden auf mehr als zwei Minuten, die das Programm damit zubringt, darauf zu warten, daß ein bestimmter Sektor in die Position unter den Lese-/Schreib-Kopf rotiert! In Anwendungen, die sehr viel Platten-Ein-/Ausgabe ausführen, wird der Zeitverlust jedoch nicht ganz so groß. Man kann nämlich etwas gegen dieses Problem unternehmen und solche Anwendungen dadurch bedeutend verbessern.

Ein Weg, das Warten auf Plattensektoren zu vermeiden, besteht darin, sie so zu verschränken, daß ein physikalischer Sektor jeweils zwei aufeinanderfolgende logische Sektoren trennt. Mit anderen Worten: Man alterniert die Sektornummern rund um die Spur. *Bild 8* zeigt, wie neun Dateistücke von der Größe eines Sektors auf einer Neun-Sektor-Spur verschränkt werden können.

Die Spuradreßtabelle erlaubt einem (über die BIOS-Funktion) festzulegen, welche logische Sektornummer jeder physikalische Sektor der Spur haben soll. Durch Festlegung der Größe eines jeden Sektors kann man die Sektorgröße rund um die Spur ändern. Die Information der Spuradreßtabelle wird auf der Platte gespeichert, so daß der Plattencontroller einen bestimmten Sektor finden kann, ohne spezielle Tabellen konsultieren zu müssen, um das Layout der Platte zu erstellen. Wenn man die Spuradreßtabelle verwendet, um das Layout der Platte festzulegen, dann erzeugt man die »Verschränkung« (interleaving) für die Spur, als permanenten Teil der logischen Struktur der PLatte.

Die Spuradreßtabelle ist eine Reihe von 4-Byte-Einträgen (einer für jeden Sektor auf der Spur), welche die Spur angeben, den Kopf, die logische Sektornummer und den Code-Umfang. *Tabelle 7* listet die erlaubten Code-Größen auf.

Die Einträge in die Spuradreßtabelle erfolgen auf der Platte immer über die physikalische Sektornummer. Jedem physikalischen Sektor wird eine logische Sektornummer (in jeder gewünschten Reihenfolge) zugeteilt, so daß die gewünschte Verschränkung festgelegt wird. Die Spuradreßtabelle legt die Zugriffs-Reihenfolge der Sektoren auf der Spur fest. PCs lesen auf der Platte, indem auf den Sektor-Header zugegriffen wird, um festzustellen, welcher Sektor gebraucht wird.

Die Spuradreßtabelle für Spur 3 einer 360-Kbyte-Diskette, doppelseitig, mit doppelter Dichte, mit neun Sektoren pro Spur, könnte so aussehen wie die in *Bild 8*. Die logischen Sektorennummern in Bild 8 entsprechen den folgenden physikalischen Sektoren:

physikalischer Sektor	1	2	3	4	5	6	7	8	9
logischer Sektor	1	6	2	7	3	8	4	9	5

Um einen Formatierfehler anzuzeigen, wird das Carry-Flag beim Verlassen der Funktion gesetzt. Wenn das Flag gesetzt ist, dann enthält AH einen Fehlercode. Die Bedeutung der einzelnen Bits dieses Fehlercodes wird in *Tabelle 8* gezeigt. Wenn ein Fehler auftritt, sollte Ihr Programm sofort BIOS Interrupt 13h, Funktion 00h aufrufen (Reset der Platte), und dann den Fehler entsprechend behandeln.

Die Funktion `fmt_trk()` ist ein in C implementiertes Beispiel für die Spurformatierprozedur (siehe *Listing 6*). Diese Funktion nimmt an, daß eine 360-Kbyte-Diskette, DSDD, auf einem PC-Standardlaufwerk formatiert werden soll. Die Funktion stellt nur eine einfache Fehlererkennung zur Verfügung; sie nimmt auch an, daß die aufrufende Routine den ganzen Dialog mit dem Benutzer abwickelt.

Zusammenfassung

In diesem Artikel haben Sie etwas über den grundlegenden Aufbau von Platten und Disketten gelernt, und wie sie formatiert werden. Und Sie

▸ *Tabelle 8: Statusbits bei Plattenfehler.*

Fehlercode Hex	Binär	Bedeutung
01	1	Falscher Befehl
02	......1.	Fehlerhafte Sektoradresse
03	11	Schreibschutzfehler
04	.....1..	Schlechter Sektor/ Sektor nicht gefunden
08	....1...	DMA-Überlauf
09	....1..1	DMA-Fehler
10	...1....	Fehlerhafte CRC-Prüfsumme beim Lesen
20	..1.....	Controller arbeitet falsch
40	.1......	Suchfehler
80	1.......	Zeit überschritten (timeout)

haben erfahren, daß das BIOS nur Spuren, Sektoren und Plattenköpfe kennt, aber keine Dateien (files) und Inhaltsverzeichnisse (directories). Das BIOS weiß, wie es die Partitionstabelle der Platte anlegen soll, und ebenso den Bootrecord für die Platte, aber sein Wissen endet hier. Neben diesen speziellen Strukturen ist der Rest der Platte nur eine Ansammlung von Daten-Sektoren.

Alle Plattenoperationen im Zusammenhang mit Dateien sind Funktionen auf DOS-Ebene. DOS verwaltet die Inhaltsverzeichnisse der Platte, die Dateien und die Dateibelegungstabellen (FATs). Struktur und Lokalisierung dieser Tabellen werden im BIOS-Parameter-Block (BPB) zur Verfügung gestellt, der im Bootrecord gespeichert wird (der erste Sektor der bootfähigen Partition).

Wenn man einmal die Basisinformation über die Platte hat, dann kann man auf weitere Information über die Platte (freier Platz, Plattenkapazität usw.) über Standard-DOS-Aufrufe zugreifen.

Terry R. Dettmann

Dieser Artikel ist ein Auszug aus dem »DOS-Programmierhandbuch, Teil 1« von Terry R. Dettmann, erschienen im Systhema-Verlag, mit dessen freundlicher Genehmigung der Abdruck erfolgt.

▸▸ *Listing 6*

```
/*
    Formatiert die spezifizierte Disk-Spur mit einem
    Standard 360K, DSDD Spur-Format. Gibt 0 zurueck, wenn
    die Formatierung erfolgreich ist; andernfalls wird ein
    Fehlercode zurueckgegeben, der von der aufrufenden
    Routine dazu verwendet werden kann, den naechsten
    Schritt festzulegen.
*/


#include <stdio.h>
#include <dos.h>

#define DISK    0x13

fmt_trk(dsk,trk,head)

int  dsk;
int  trk;
int  head;

{
     union REGS regs;
     char trktbl[36];
     int  i;

     for(i=0; i< 9; i++){
          trktbl[ i*4]   = trk;
          trktbl[ i*4+1] = head;
          trktbl[ i*4+2] = i;
          trktbl[ i*4+3] = 2;
     }
     regs.h.ah = 0x05;
     regs.h.ch = trk;
     regs.h.dh = head;
     regs.h.dl = dsk;
     regs.x.bx = trktbl;
     int86(DISK,&regs,&regs);
     if(regs.x.cflag)
          return(fmt_error(regs.h.ah));
     return(0);
}

fmt_error(code)


/*
 Verarbeitet einen Disk-Formatierfehler durch Reset des
 Laufwerks und Rueckgabe eines Fehlercodes. Der Code, der
 an diese Routine uebergeben wird, ist ein Bitcode, der so
 interpretiert wird, wie es in Tabelle 8 gezeigt wird.
 Diese Routine gibt einen Fehlercode 1 zurueck, um
 anzuzeigen, dass ein "Schreibschutz"-Fehler (write-protect
 error) aufgetreten ist (der festgestellt werden kann).
 Die Routine nimmt an, dass alle anderen Fehler nicht
 unterschieden werden koennen; deshalb werden sie alle
 zusammengeworfen.
*/

char code;

{
     union REGS regs;

     regs.h.ah = 0;
     int86(DISK,&regs,&regs);
     return((code==3)?1:2);
}

/* ===================================================== */
```

Speicherbeschränkungen unter DOS elegant unmgehen:

Ein Overlay-Manager für DOS

Zur Zeit stehen PC-Anwendern und Programmierern eine sehr große Anzahl von Software-Werkzeugen zur Verfügung. MS-DOS- und PC-DOS-Systeme können durch tausende von speicherresidenten (TSR-)Routinen um sinnvolle und zeitsparende Funktionen erweitert werden, integrierte Arbeitsumgebungen und Multitasking-Oberflächen steigern die Leistungsfähigkeit von PCs. Ein großer Nachteil besteht allerdings im zum Teil durchaus beträchtlichem Speicherbedarf dieser Programme. Entwickler müssen darum Ihre Anwendungen sauber strukturieren und dem Design des Programmcodes große Aufmerksamkeit schenken, um durch die Benutzung von kleineren Funktionsmodulen das mehrfache Auftreten von gleichen Programmsequenzen zu verringern.

Da der Umfang des Programmcodes zwar trotz aller Optimierung stark reduziert werden kann, reicht das in manchen Fällen immer noch nicht aus, die Anwendung ist für den zur Verfügung stehenden Speicher immer noch zu groß. Dann ist ein Overlay-Manager vielleicht genau die richtige Lösung des Problems.

Die Verwendung von Overlay-Konzepten ist ein durchaus übliches Verfahren, um Datenspeicher besser auszunutzen. C-Programmierern stehen die Funktionen malloc und free, Pascal-Programmierern die Funktionen New, Free, Mark und Release zur Verfügung, um Speicherbereiche dynamisch verwalten zu können. Die grundlegende Idee eines Overlay-Managers besteht darin, daß sich nicht alle Routinen eines Programms zur gleichen Zeit im Speicher befinden müssen.

In einem Textverarbeitungsprogramm sind z.B. die Routinen zur Druckausgabe selten aktiv, wenn das Dokument bearbeitet wird. Overlays ermöglichen es, die Druckroutinen solange auf der Festplatte zu belassen, bis sie wirklich benötigt werden. Dann erst ist es notwendig, den Programmcode in den Speicher zu laden und die Editierfunktionen auf die Festplatte zu schreiben, um ihn später erneut zu lesen oder zu löschen. Dadurch benötigt das Textverarbeitungsprogramm deutlich weniger Hauptspeicher.

Ein anderes Beispiel ist die Benutzung von mehreren Druckertreibern, von denen zu einem bestimmten Zeitpunkt lediglich einer geladen sein muß, da immer nur ein Drucker in Betrieb ist.

Benutzung von Overlays

Der einzige vernünftige Grund für die Benutzung von Overlays ist die Minimierung von benötigtem Speicher für eine Anwendung, da sich die Ausführungszeit des Programms durch die notwendigen Plattenzugriffe erhöht, aber diese Minimierung kann überaus wichtig sein.

Die Arbeit eines Programmierers ist, unabhängig von der verwendeten Overlay-Technik, viel einfacher, wenn der Programmcode nicht wesentlich verändert werden muß. Insbesondere sollte die Anwendung gut strukturiert und wie gewohnt »eindimensional« entwickelt und getestet werden, so daß das Aufsetzen einer Overlay-Verwaltung als Abschluß der Entwicklung möglich ist, denn das Austesten und die Beseitigung von Fehlern aus einem Programm mit Overlays kann unter Umständen mehr als nervenaufreibend sein. Ebenso sollten die Programmteile für Overlays erst so spät im Entwicklungszyklus bestimmt werden, wenn sowohl die Komplexität, als auch die Größe der Routinen erkennbar ist oder sogar feststeht.

▶ *Listing 1: OVRMGR.ASM.*

```
                .model  LARGE
                extrn   $$INTNO:byte
                extrn   $$MPGSNBASE:word
                extrn   $$MPGSNOVL:byte
                extrn   $$MPOVLLFA:word
                extrn   $$OVLBASE:byte

                extrn   $$MAIN:far
                extrn   _read_overlay_section:far
                extrn   _errputs:far

                .data
_pspaddr        dw      ?               ;for relocation with read_overlay_section
old_ovint       dw      ?               ;old overlay interrupt offset
                dw      ?               ;and segment
_executable_name        db      80 dup (0)      ;space for full pathname

cantmap         db      'Can''t map overlay!...exiting',13,10,0
reentry         db      'Already mapping an overlay!...exiting',13,10,0


                .code

                public  $$OVLINIT
                public  _executable_name
                public  _pspaddr
                public  ax_save,bx_save,cx_save,es_save,ret_ip,ret_cs
                public  req_ov,ov_ofs,ov_seg,in_ovly_mgr,ovly_int,return_from_ov

ax_save         dw      ?               ;save area for used registers
bx_save         dw      ?
cx_save         dw      ?
es_save         dw      ?

ret_ip          dw      ?               ;original return address
ret_cs          dw      ?               ;(from call to overlaid routine)

req_ov          db      ?               ;requested overlay number
ov_ofs          dw      ?               ;address to call after mapping
ov_seg          dw      ?

in_ovly_mgr     db      0               ;flag to indicate already mapping

$$OVLINIT       label   far
                push    ax
                push    bx
                push    dx
                push    es
                push    ds                      ;need to use ds:dx
                mov     ax,DGROUP
                mov     ds,ax

                mov     ax,es                   ;initialize _pspaddr
                mov     _pspaddr,ax

                mov     al,[$$INTNO]            ;get interrupt number
                mov     ah,35h                  ;get overlay number interrupt vector
                int     21h

                mov     old_ovint,bx
                mov     old_ovint+2,es

                mov     bx,cs
                mov     ds,bx
                mov     dx,offset ovly_int      ;new interrupt
                mov     ah,25h
                int     21h                     ;install new int handler
                pop     ds
                pop     es
                pop     dx
                pop     bx
                pop     ax

                jmp     $$MAIN                  ;go to mainline code


;**
;** INTERRUPT HANDLER
;**

ovly_int        proc    far
                cmp     cs:in_ovly_mgr,0        ;are we not here already?
                je      ok                      ;right
; Error if we're already in overlay manager!
                mov     ax,offset reentry       ;error message
                push    ds
                push    ax
                call    _errputs
                add     sp,4
                mov     ax,4c41h                ;exit with error code 41h
                int     21h

ok:             mov     cs:in_ovly_mgr,1        ;note that we're here
                mov     cs:ax_save,ax
                mov     cs:bx_save,bx
                mov     cs:cx_save,cx
                mov     ax,es
                mov     cs:es_save,ax

                pop     bx                      ;bx = return ip
                push    bx
                add     bx,3                    ;adjust for extra crap
                mov     cs:[ret_ip],bx          ;save it
                pop     bx                      ;get original retadd back

                pop     ax                      ;return cs
                mov     cs:[ret_cs],ax          ;save it
                mov     es,ax                   ;es:bx -> bytes after INT

                pop     ax                      ;flags, discard

                xor     ah,ah
                mov     al,byte ptr es:[bx]     ;get requested overlay number
                inc     bx                      ;move to next item
                mov     byte ptr cs:req_ov,al   ;save it
                mov     cx,word ptr es:[bx]     ;get offset in overlay segment
                mov     cs:[ov_ofs],cx          ;save it

                push    ax                      ;p3 for C function
                mov     bx,ax
                shl     bx,1                    ;* 2
                add     bx,offset $$MPGSNBASE   ;add to base of segment table
                mov     bx,[bx]
                mov     ov_seg,bx
                push    bx                      ;push the segment for p2
                mov     ax,0                    ;p2, low word (0)
                push    ax
                call    _read_overlay_section   ;call the pup!
                add     sp,6                    ;get rid of parms
                cmp     ax,0                    ;error return?
                je      no_error                ;no
```

Diese Strukturierung kann das Aufsetzen der Overlay-Verwaltung zu einem späten Zeitpunkt der Programmentwicklung derart vereinfachen, daß ein Großteil der sonst notwendigen Programmänderungen einfach wegfällt.

Wie die meisten Entwickler und Anwender sicher schon festgestellt haben, erleichtern TSR-Routinen die Arbeit mit dem PC zum Teil ganz erheblich. Texteditoren, Kalender, Weckfunktionen, Terminkalender, Hilfefunktionen und Dokumentationen, Kommunikationsprogramme und nicht zuletzt Spiele sind zwar »nur einen Tastendruck entfernt«, belegen aber auch wertvollen Speicherplatz. Ohne Schwierigkeiten können zwei oder drei zusätzlich geladene TSR-Routinen den Arbeitsbereich eines Texteditors oder Compilers stark einschränken. Aus diesem Grund ist ein Overlay von speicherresidenten Programmen empfehlenswert, insbesondere, da die TSR-Routinen die meiste Zeit über sowieso inaktiv sind. Erst bei einer Aktivierung muß Speicherplatz vom DOS belegt und der Programmcode geladen werden. Nach der Deaktivierung wird dieser Speicherplatz wieder freigegeben und steht anderen Anwendungen zur Verfügung.

Vorteilhaft ist auch die Benutzung von TSR-Routinen, die nicht benötigte Programmteile von bereits aktiven, komplexen Anwendungen überlagern können. Zum Beispiel belegt ein Textverarbeitungsprogramm mit Funktionalitäten für Desktop-Publishing ohne weiteres 500 Kbyte und blockiert fast den gesamten Hauptspeicher. Wenn Overlay-Techniken verwendet werden, läßt sich der benötigte Speicherbedarf unter Umständen auf 300 Kbyte reduzieren und damit Platz für weitere Routinen schaffen. Die Laufzeit der Anwendung erhöht sich dadurch zwar, aber durch ein durchdachtes Design der einzelnen Overlay-Strukturen kann der Entwickler das Ein- und Auslagern von Speicherblöcken zwischen Primär- und Sekundärspeicher (Swapping) minimieren und vielleicht sogar Zeiten nutzen, die ein Benutzer zum Lesen von Bildschirm-Meldungen benötigt. Die Verringerung der Programmgröße wiegt mitunter auch den Nachteil einer deutlich erhöhten Laufzeit wieder auf. Um die auftretenden Zeitprobleme ein wenig abzuschwächen, kann der überlagerte Programmcode in einen LIM/EMS-Speicher gespeichert werden, dessen Zugriffszeiten weit unter denen von Festplatten liegen.

Auch wenn es nicht gerade einfach ist, Programme aus dem EMS-Bereich ablaufen zu lassen (vor allem in älteren EMS-Versionen), ist die Benutzung von Expanded Memory vorteilhaft für eine Overlay-Verwaltung, denn das Kopieren von Programmblöcken zwischen dem EMS-Bereich und der Overlay-Verwaltung ist relativ unkompliziert.

Das gleiche Ziel kann mit einem Platten-Cache im EMS erreicht werden, wenn auch mit einer deutlich geringeren Effizienz.

◂ *Listing 1: (Ende)*

```
; Error if can't map this overlay!
                mov     ax,offset cantmap       ;error message
                push    ds
                push    ax
                call    _errputs
                add     sp,4
                mov     ax,4c42h                ;exit with error code 42h
                int     21h

no_error:       push    cs
                mov     ax,offset return_from_ov;trick up return to our code
                push    ax
                mov     ax,seg $$OVLBASE        ;set up call to overlay code
                push    ax
                mov     ax,ov_ofs
                push    ax
                retf                            ;jump to overlay code...

return_from_ov:                                 ;..and come back here
                mov     ax,ret_cs               ;restore original retadd
                push    ax
                mov     ax,ret_ip
                push    ax
                mov     ax,cs:es_save           ;and original reg values
                mov     es,ax
                mov     ax,cs:ax_save
                mov     bx,cs:bx_save
                mov     cx,cs:cx_save
                mov     cs:in_ovly_mgr,0        ;we're not here anymore
                ret
ovly_int        endp

                end
```

Erste Ansätze

Schon seit einer geraumen Zeit wird an der Entwicklung von verschiedenen Techniken zur Implementierung von Overlays gearbeitet. Einen sehr frühen Lösungsansatz stellt die Einbettung von Sprachelementen als Anweisungen in Hochsprachen, z.B. OVERLAY FUNCTION oder OVERLAY SUBROUTINE, dar. Auf einigen Großrechnern stehen in FORTRAN-Compilern diese Sprachelemente zur Verfügung, mit denen der Overlay-Prozeß gesteuert werden kann. Ein weiterer Ansatz waren vom Entwickler nicht zu beeinflussende System-Overlays, wie sie WordStar (in CP/M- und DOS-Version) oder Lotus 1-2-3 bieten.

Die wesentlichen Nachteile dieser beiden Techniken bestehen in der Komplexität des Programmcodes und der fehlenden Transparenz für den Entwickler. Durch zusätzliche Arbeiten werden darüber hinaus auch unnötige Fehler erzeugt, und die sind in Overlay-Systemen sehr schwer zu lokalisieren. Mit anderen Ansätzen der Overlay-Verwaltung erhalten Entwickler mehr Freiraum beim Design von Programmcode, da das Betriebssystem die Kontrolle des Swapping übernimmt.

Früher wurden auf Großrechnern z.B. ganze Prozesse geswappt, d.h. mehrere Anwendungen teilten sich die Rechnerkapazitäten, indem die Programme als Ganzes auf den Sekundärspeicher ausgelagert wurden und danach die Kontrolle an das nachfolgende Programm überging.

Moderne Algorithmen unterteilen Anwendungen für das Swapping in kleinere Einheiten als Prozesse oder Programme, in sogenannte Seiten (engl. pages, segments). Diese Techniken bezeichnet man als Demand-Paging bzw. (segmentierte) virtuelle Speicherverwaltung (s. dazu auch Microsoft System Journal, Ausgabe Sept./Okt. 89 »Eine virtuelle Speicherverwaltung in C«).

Beim Ein- und Auslagern von Programmblöcken sind dadurch höhere Feinheitsgrade der Aufteilung von Anwendungen möglich als beim Swapping von ganzen Prozessen, der Speicherplatz wird besser ausgenutzt und I/O auf die Platten reduziert. Allerdings entstehen auch Nachteile, wenn das Betriebssystem das Ein- und Auslagern von Programmblöcken selbsttätig steuert. Zum einen ist ein erhöhter Aufwand zur Speicherverwaltung notwendig, da nicht alle Segmente eines Prozesses sofort zur Verfügung stehen. Außerdem ist der verwendete Swapping-Algorithmus auf einen Idealfall ausgelegt, der so natürlich niemals vorkommt. Anwendungen sollten individuell segmentiert werden, damit auch zusammenhängende Seiten gleichzeitig im Hauptspeicher geladen werden. Diese, vom jeweiligen Anwendungsfall abhängige Strukturierung, ist von einem allgemeingültigen Algorithmus nicht zu leisten. Darüber hinaus arbeitet der Swapping-Algorithmus des Betriebssystems unabhängig von der Programmlogik. Daß z.B. die Initialisierungsroutinen einer Anwendung nur einmal pro Programmlauf benötigt werden, ist für den Algorithmus natürlich überhaupt nicht ersichtlich.

Aktuelle Ansätze

PC-Entwicklern, die Overlay-Techniken auf ihre Programme anwenden wollen, stehen heute verschiedene, neue Lösungsansätze zur Verfügung. PLINK, ein Linker von Phoenix Technologies bietet eine sehr hohe Flexibilität für Daten- und Programm-Overlays und arbeitet mit Microsoft .OBJ-Dateien. Allerdings ist PLINK kein Standard-Linker und die Einarbeitung erfordert unter Umständen einige Zeit.

Die Microsoft-Compiler für C und FORTRAN ermöglichen Overlay-Techniken mit Hilfe des Microsoft Overlay-Linkers LINK und Overlay-Verwaltungen, die im Sprachumfang, d.h. der Run-Time-Bibliothek, enthalten sind. Eine Ausnahme bildet hier der Microsoft Macro-Assembler MASM, der die Möglichkeiten des Linkers für Overlays nur im Mixed-Language-Modell mit Hochsprachen nutzen kann. Außerdem unterstützen der Microsoft Overlay-Manager und der Linker keine Sprachen, die von Microsoft nicht angeboten werden, z.B. C++, Modula-2 und PL/I. Ebenfalls ausgenommen ist Microsoft QuickBASIC.

Der Linker

LINK bietet Möglichkeiten für Overlay-Techniken auch den Entwicklern, die nicht in Microsoft C oder FORTRAN programmieren, und zwar in einer sehr einfachen Form. Ein Programmteil wird als Overlay-Bereich definiert, wobei bis zu 255 verschiedene Codesegmente als Overlay festgelegt werden können (der Overlay-Bezeich-

▸ *Listing 2: RDOVLY.C.*

```
#include "exehdr.h"

/* externally-defined full pathname of executable */

#define min(a,b) ((a) < (b)) ? (a) : (b)
#define max(a,b) ((a) < (b)) ? (a) : (b)
#define OPENMODE_RO 0
#define MK_FP(a,b) ((void far *)( ((unsigned long)(a) << 16) | (unsigned)(b) ))

extern char executable_name[];
extern int pspaddr;

void errputs(char far *s);
long mylseek(int handle, long offset, int type);
int myread(int handle, void far *buf, int len);
int myopen(char far *name, int mode);
int myclose(int handle);


int read_overlay_section(
void far *ovarea,                   /* buffer into which to read him */
int requested_overlay_number        /* number of overlay to read */
)

#define RELOC_BUF_SIZE 32
{
    struct reloc_entry_ reloc_buf[RELOC_BUF_SIZE];
    struct reloc_entry_ *reloc_ptr;
    int reloc_chunk;
    unsigned far *target_ptr;
    struct exehdr_ eh;
    int i;
    long startpos, nextpos, filesize;
    long sectionsize, imagesize;
    int executable_handle;

    /* open executable */
    executable_handle = myopen(executable_name,OPENMODE_RO);

    /* determine file size */
    filesize = mylseek(executable_handle,0L,2);

    /* search from beginning of file */
    mylseek(executable_handle,0L,0);

    while (1) {
        /* use mylseek() to avoid calling runtime functions */
        startpos = mylseek(executable_handle,0L,1);
        if (myread(executable_handle,&eh,sizeof(eh)) != sizeof(eh)) {
            errputs("Something's wrong...can't read .EXE header\r\n");
            myclose(executable_handle);
            return(1);
      }

         if (eh.M_sig != 'M' || eh.Z_sig != 'Z') {
            errputs("Found non-EXE signature!\r\n");
            myclose(executable_handle);
            return(1);
            }
        if (eh.remain_len == 0)
            sectionsize = (long)eh.page_len * 512;
        else
            sectionsize = (long)(eh.page_len - 1) * 512 + eh.remain_len;

        if (eh.overlay_number == requested_overlay_number) {
            /* found ours...load and fix up */

            /* move to executable image */
            mylseek(executable_handle, startpos + eh.hsize * 16, 0);
            myread(executable_handle, ovarea, (int)(sectionsize - eh.hsize * 16));

            /* fix up relocations in the loaded overlay */

            mylseek(executable_handle,startpos + (long)eh.first_reloc,0);
            while (eh.num_relocs) {
                reloc_chunk = min(RELOC_BUF_SIZE,eh.num_relocs);
                myread(executable_handle,reloc_buf,reloc_chunk
                                              * sizeof(struct reloc_entry_));
                eh.num_relocs -= reloc_chunk;
                for (i = 0; i < reloc_chunk; i++) {
                    reloc_ptr = reloc_buf + i;
                    target_ptr = MK_FP(pspaddr + 0x10 + reloc_ptr->segment,
                                 reloc_ptr->offset);
                    *target_ptr += pspaddr + 0x10;
                }
            }
            myclose(executable_handle);
            return 0;
        } else {
        nextpos = startpos + sectionsize;
            /* round up to 512-byte bound */
            nextpos = (nextpos + 511L) & ~511L;
        if (nextpos >= filesize) {
                myclose(executable_handle);
            return 1;
            }
        mylseek(executable_handle,nextpos,0);
        }
    }
    myclose(executable_handle);
    return 1;
}
```

ner hat die Größe 1 Byte). Jedes Overlay-Segment besteht aus einem oder mehreren Modulen (Objekt-Dateien), von denen sich zu einem bestimmten Zeitpunkt jeweils eines im Overlay-Bereich befindet.

Der Overlay-Manager wird ähnlich einer Interruptroutine installiert: Der Linker verlagert die Anfangsadresse des Programms, den sogenannten Entry-Point, auf einen bestimmten Bereich innerhalb der Overlay-Verwaltung, so daß der Manager sich selbst initialisieren kann, bevor das eigentliche Programm ausgeführt wird. Danach übersetzt der Linker alle Aufrufe von Overlay-Routinen in Software-Interrupts, damit der Overlay-Manager den Overlay-Code von der Platte laden kann, bevor die Programmkontrolle an die gerufene Routine übergeht. Das wird weiter unten noch ausführlich beschrieben. Es befindet sich zwar zu einem bestimmten Zeitpunkt nur ein Segment im Overlay-Bereich, dieses Segment kann aber mehrere Objekt-Module beinhalten. Zum Beispiel wird durch das Kommando

```
LINK main+(p1+p2)+(p3);
```

durch den Linker eine Datei MAIN.EXE erzeugt, die aus dem residenten Programmteil main und zwei Overlay-Segmenten besteht. Dabei bilden die Module p1, p2 und p3 jeweils ein Segment.

Zu beachten ist hierbei aber in jedem Fall, daß die Module p1, p2 und p3 mit dem Klassennamen `'CODE'` bezeichnet sind und alle Module als eigenständige Segmente definiert sind, die sich alle von dem Segment in main unterscheiden. In den hier aufgeführten Beispielen wird das durch die .LARGE-Direktive von MASM und die Verwendung separater Quelldateien gewährleistet. Bei einer anderen Codierung als in diesen Beispielen sollte dieser Formalismus aber genauso eingehalten werden.

Da die Routinen ähnlich Software-Interrupts über FAR-Adressen referiert werden, ist die Verwendung des FAR-Modells für den Overlay-Manager unbedingt notwendig. Der Overlay-Linker kann somit die FAR-Referenz auf die Routine (5 Byte) durch einen Interruptaufruf (2 Byte), eine Overlay-Nummer (1 Byte) und einen Offset innerhalb des Overlay-Bereiches (2 Byte) ersetzen. Der Offset ist insbesondere dann notwendig, wenn das Overlay mehr als ein Modul enthält. Zum Beispiel kann die Anweisung CALL MY_ROUTINE durch INT 3FH (den Default-Interrupt für Overlays) DB 01 (als Bezeichnung für das erste Overlay) DW 0100 (die Routine beginnt bei Offset 100H im Overlay-Bereich) ersetzt werden.

LINK verändert dabei aber nur die Aufrufe, die sich auf Overlay-Routinen beziehen. Interessanterweise wird ein mit Overlays gebundenes Programm durch SYMDEB auch als

```
INT 3FH; <Overlay-Nummer> <Overlay-Offset>
```

disassembliert. SYMDEB kann zumindest teilweise Overlay-Code erkennen. Der Debugger CodeView von Microsoft besitzt darüber hinaus auch bestimmte Funktionen, die das Arbeiten mit lokalen Symbolen in Overlay-Routinen ermöglichen.

Diese Technik des Overlay-Managers behandelt allerdings durch Zeiger referierte Funktionen (indirekte Adressierung) nicht korrekt. Wird also der hier beschriebene Manager auf eine Overlay-Funktion angewendet, die nur über einen Zeiger erreicht werden kann, ersetzt der Linker den Funktionsaufruf nicht durch die Anweisung INT 3FH und der Overlay-Manager kann

die Routine nicht laden. Das wirkt sich natürlich fatal auf den Programmablauf aus. Außerdem wird eine .EXE-Datei mit Overlay-Strukturen durch LINK anders als gewöhnlich aufgebaut. Der Linker erzeugt für jedes Overlay-Segment einen eigenen EXE-Kopf, der besondere Felder für die Overlay-Routine sowie einen Datenblock ausschließlich für das Codesegment enthält. Diese Strukturen werden im folgenden noch genauer untersucht.

Globale Variablen

Der Linker erweitert die Module um einige, vom Overlay-Manager benutzte, globale Variablen:

• OVERLAY_AREA, OVERLAY_END sind zwei Segmentnamen, die am Ende des CODE-Segments vom Linker definiert werden. OVERLAY_AREA bezeichnet den Beginn des Overlay-Bereichs, OVERLAY_END den ersten freien Paragraphen hinter dem größten Overlay, somit das Ende des Overlay-Bereichs.

• $$CGSN diese Variable enthält die Anzahl voneinander unabhängiger Overlay-Segmente. In unserem Overlay-Manager wird sie nicht benutzt, da dieser nur die .EXE-Datei nach dem nächsten angeforderten Overlay durchsucht.

• $$COVL enthält den gleichen Wert wie $$CGSN und bezeichnet die maximale Anzahl von Overlays. $$CGSN ist Parameter für die Größe des Feldes $$MPGSNBASE.

• $$EXENAM identifiziert das ausführbare Programm durch Dateinamen und Erweiterung, also .EXE. In diesem Overlay-Manager wird sie nicht verwendet, da der vollständige Dateiname im DOS-Environment des Programms enthalten ist.

• $$INTNO ist die Nummer des Software-Interrupts, den der Overlay-Manager benutzt. Durch den Steuerparameter /OVERLAYINTERRUPT von LINK läßt sich der Defaultwert 3FH verändern. Der Overlay-Manager sollte diesen Variablenwert übernehmen und mit dem zugehörigen Interruptvektor die Aufrufe der Overlay-Routinen abfangen.

• $$MAIN ist die ursprüngliche Startadresse des Programms, zu der verzweigt würde, wenn keine Overlays existierten. Wenn das Programm durch Overlays strukturiert ist, enthält $$OVLINIT die neue Startadresse (s.u.). Nach der Initialisierung des Overlay-Managers und der Übernahme des Interruptvektors $$INTNO kann die Programmkontrolle auf $$MAIN übergehen.

• $$MPGSNBASE bezeichnet ein Feld, das aus $$CGSN-Worten besteht, wobei jedes Wort die Segmentadresse eines Overlays enthält. Das Wort $$MPGSNBASE[0] enthält die Segmentadresse des residenten Teils des Programms, also die Adresse des PSP (Program Segment Prefix) + 10H. In die folgenden $$CGSN-1 Feldelemente ist jeweils der Wert OVERLAY_SEGMENT eingetragen. Mit $$MPGSNBASE lassen sich die

◄ *Listing 3: RDOVLY.ASM.*

```
;       Static Name Aliases
;
        TITLE   rdovly.c
        NAME    rdovly

        .8087
RDOVLY_TEXT     SEGMENT  WORD PUBLIC 'CODE'
RDOVLY_TEXT     ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP   CONST, _BSS, _DATA
        ASSUME  CS: RDOVLY_TEXT, DS: DGROUP, SS: DGROUP
EXTRN   _myread:FAR
EXTRN   _myopen:FAR
EXTRN   _myclose:FAR
EXTRN   _errputs:FAR
EXTRN   _mylseek:FAR
EXTRN   _executable_name:BYTE
EXTRN   _pspaddr:WORD
_DATA      SEGMENT
$SG156  DB      'Something''s wrong...can''t read .EXE header',  OdH,  OaH,  OOH
$SG159  DB      'Found non-EXE signature!',  OdH,  OaH,  OOH
_DATA      ENDS
RDOVLY_TEXT      SEGMENT
        ASSUME  CS: RDOVLY_TEXT
;#include "exehdr.h"
;
;/* externally-defined full pathname of executable */
;
;#define min(a,b) ((a) < (b)) ? (a) : (b)
;#define max(a,b) ((a) < (b)) ? (a) : (b)
;#define OPENMODE_RO 0
;#define MK_FP(a,b) ((void far *)( ((unsigned long)(a) << 16) | (unsigned)(b) ))
;
;extern char executable_name[];
;extern int pspaddr;
;
;void errputs(char far *s);
;long mylseek(int handle, long offset, int type);
;int myread(int handle, void far *buf, int len);
;int myopen(char far *name, int mode);
;int myclose(int handle);
;
;
;int read_overlay_section(
;void far *ovarea,                     /* buffer into which to read him */
;int requested_overlay_number          /* number of overlay to read */
;)
;
;#define RELOC_BUF_SIZE 32
;{
        PUBLIC  _read_overlay_section
_read_overlay_section   PROC FAR
        push    bp
        mov     bp,sp
        sub     sp,190
        push    di
        push    si
;       ovarea = 6
;       requested_overlay_number = 10
;       reloc_buf = -140
;       reloc_ptr = -146
;       reloc_chunk = -186
;       target_ptr = -154
;       eh = -184
;       i = -142
;       startpos = -8
;       nextpos = -12
;       filesize = -150
;       sectionsize = -4
;       imagesize = -190
;       executable_handle = -156
;    struct reloc_entry_ reloc_buf[RELOC_BUF_SIZE];
;    struct reloc_entry_ *reloc_ptr;
;    int reloc_chunk;
;    unsigned far *target_ptr;
;    struct exehdr_ eh;
;    int i;
;    long startpos, nextpos, filesize;
;    long sectionsize, imagesize;
;    int executable_handle;
;
;    /* open executable */
;    executable_handle = myopen(executable_name,OPENMODE_RO);
        mov     ax,0
        push    ax
        mov     ax,OFFSET _executable_name
        mov     dx,SEG _executable_name
        push    dx
        push    ax
        call    FAR PTR _myopen
        add     sp,6
        mov     WORD PTR [bp-156],ax    ;executable_handle
;
;    /* determine file size */
;    filesize = mylseek(executable_handle,0L,2);
        mov     ax,2
        push    ax
        sub     ax,ax
        push    ax
        push    ax
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _mylseek
        add     sp,8
        mov     WORD PTR [bp-150],ax    ;filesize
        mov     WORD PTR [bp-148],dx
;
;    /* search from beginning of file */
;    mylseek(executable_handle,0L,0);
        mov     ax,0
        push    ax
        sub     ax,ax
        push    ax
        push    ax
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _mylseek
        add     sp,8
;
;    while (1) {
$FC153:
;        /* use mylseek() to avoid calling runtime functions */
;        startpos = mylseek(executable_handle,0L,1);
        mov     ax,1
        push    ax
        sub     ax,ax
        push    ax
        push    ax
```

▸ *Listing 3: (Fortsetzung)*

```
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _mylseek
        add     sp,8
        mov     WORD PTR [bp-8],ax      ;startpos
        mov     WORD PTR [bp-6],dx
;         if (myread(executable_handle,&eh,sizeof(eh)) != sizeof(eh)) {
        mov     ax,28
        push    ax
        lea     ax,WORD PTR [bp-184]    ;eh
        push    ss
        push    ax
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myread
        add     sp,8
        cmp     ax,28
        jne     $JCC132
        jmp     $I155
$JCC132:
;               errputs("Something's wrong...can't read .EXE header\r\n");
        mov     ax,OFFSET DGROUP:$SG156
        push    ds
        push    ax
        call    FAR PTR _errputs
        add     sp,4
;               myclose(executable_handle);
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myclose
        add     sp,2
;               return(1);
        mov     ax,1
        jmp     $EX139
;           }
;
;           if (eh.M_sig != 'M' || eh.Z_sig != 'Z') {
$I155:
        cmp     BYTE PTR [bp-184],77    ;eh
        je      $JCC173
        jmp     $I158
$JCC173:
        cmp     BYTE PTR [bp-183],90
        jne     $JCC183
        jmp     $I157
$JCC183:
$I158:
;               errputs("Found non-EXE signature!\r\n");
        mov     ax,OFFSET DGROUP:$SG159
        push    ds
        push    ax
        call    FAR PTR _errputs
        add     sp,4
;               myclose(executable_handle);
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myclose
        add     sp,2
;               return(1);
        mov     ax,1
        jmp     $EX139
;           }
;         if (eh.remain_len == 0)
$I157:
        cmp     WORD PTR [bp-182],0
        je      $JCC224
        jmp     $I160
$JCC224:
;               sectionsize = (long)eh.page_len * 512;
        mov     ax,WORD PTR [bp-180]
        sub     dx,dx
        shl     ax,1
        rcl     dx,1
        mov     dh,dl
        mov     dl,ah
        mov     ah,al
        sub     al,al
        mov     WORD PTR [bp-4],ax      ;sectionsize
        mov     WORD PTR [bp-2],dx
;         else
        jmp     $I161
$I160:
;               sectionsize = (long)(eh.page_len - 1) * 512 + eh.remain_len;
        mov     ax,WORD PTR [bp-180]
        dec     ax
        sub     dx,dx
        shl     ax,1
        rcl     dx,1
        mov     dh,dl
        mov     dl,ah
        mov     ah,al
        sub     al,al
        add     ax,WORD PTR [bp-182]
        adc     dx,0
        mov     WORD PTR [bp-4],ax      ;sectionsize
        mov     WORD PTR [bp-2],dx
$I161:
;
;         if (eh.overlay_number == requested_overlay_number) {
        mov     al,BYTE PTR [bp-158]
        sub     ah,ah
        cmp     ax,WORD PTR [bp+10]     ;requested_overlay_number
        je      $JCC297
        jmp     $I162
$JCC297:
;               /* found ours...load and fix up */
;
;               /* move to executable image */
;               mylseek(executable_handle, startpos + eh.hsize * 16, 0);
        mov     ax,0
        push    ax
        mov     ax,WORD PTR [bp-176]
        mov     cl,4
        shl     ax,cl
        sub     dx,dx
        add     ax,WORD PTR [bp-8]      ;startpos
        adc     dx,WORD PTR [bp-6]
        push    dx
        push    ax
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _mylseek
        add     sp,8
;
;               myread(executable_handle, ovarea, (int)(sectionsize - eh.hsize * 16));
        mov     ax,WORD PTR [bp-4]      ;sectionsize
        mov     dx,WORD PTR [bp-176]
        mov     cl,4
        shl     dx,cl
        sub     ax,dx
        push    ax
        push    WORD PTR [bp+8]
        push    WORD PTR [bp+6] ;ovarea
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myread
        add     sp,8
```

Adressen der angeforderten Overlay-Segmente bestimmen.

- $$MPOVLLFA stellt einen Vorrat an Offsets in Overlay-Dateien des Microsoft Overlay-Managers dar. Es sind genauso viele Einträge von Doppelworten vorhanden, wie Overlays für das Programm existieren. Die Feldelemente werden mit 0 initialisiert, d.h. enthält $$MPOVLLFA[n] einen Wert ungleich 0, ist dieser Wert genau der Offset in die Datei mit dem Overlay n.

In dem hier beschriebenen Overlay-Manager wird dieses Feld nicht benutzt, obwohl damit größere Anwendungen deutlich beschleunigt werden können. Für kleinere Anwendungen ist der Zeitvorteil allerdings minimal.

- $$OVLBASE entspricht der Adresse OVERLAY_SEGMENT. Die Benutzung dieses vordefinierten Symbols stellt eine gleichwertige Alternative zur Variablen OVERLAY_SEGMENT dar, ein Overlay an die richtige Stelle zu laden. In unserem Overlay-Manager wird $$OVLBASE benutzt.
- $$OVLINIT enthält die neue Startadresse des Programms mit Overlay-Strukturen. Der Overlay-Manager sollte $$OVLINIT auf seine eigene Startadresse setzen und nach der Initialisierung zu $$MAIN verzweigen. $$MAIN ist vom Linker mit der ursprünglichen Startadresse des Programms gesetzt worden.

Struktur der .EXE-Datei

Wie oben schon erwähnt, unterscheidet sich eine .EXE-Datei, die gemeinsam mit Overlays gebunden wurde, in ihrer Struktur ein wenig von einem normalen, ausführbaren DOS-Programm. Jedes Overlay-Segment, also die Module, die beim Binden durch LINK in Klammern zusammengefaßt werden, besitzt seinen eigenen EXE-Kopf. Dieser Kopf ist ähnlich dem gewöhnlichen EXE-Kopf aufgebaut, wobei einige Daten, die nur vom DOS-Loader benötigt werden und sich auf das Gesamtprogramm beziehen, nicht enthalten sind. Werte für den initialisierten Stack, Startadresse des Programms und andere Daten erscheinen also nur im Kopf des Programm-Hauptteils. Lediglich Overlay-Nummern und Längenbezeichner zum Auffinden des nächsten Programmabschnitts unterscheiden sich in jedem Overlay-Segment. Die Kopfstruktur ist vollständig in der Datei EXEHDR.H beschrieben. Es ist natürlich wie bei jedem LINK-Lauf durchaus sinnvoll, sich mit der Option /M eine .MAP-Datei zu erzeugen.

Der Overlay-Manager

Wenn man die Funktionalitäten von LINK kennt, kann man einen einfachen Overlay-Manager entwickeln, der auch die Sprachen unterstützt,

die eigentlich keine Overlay-Strukturen ermöglichen, wie z.B. MASM. Die Programme OVRMGR.ASM (*Listing 1*) und RDOVLY.C (*Listing 2*) bilden zusammen einen Overlay-Manager, der sowohl MASM, als auch Sprachen, die nicht von Microsoft stammen, unterstützt. OVRMGR.ASM beinhaltet die Initialisierung der Overlay-Verwaltung und die Umgebung für die Handhabungsroutinen der Interrupts. RDOVLY.C liefert den ganzen Rest, z.B. das Lesen des ausführbaren Codes und das Laden der Overlay-Routinen auf die Adressen, die OVRMGR.ASM vorgibt. Die Codierung von Funktionen auf diesem Abstraktionsniveau, und damit verbunden natürlich auch die Fehlersuche, ist mit C wesentlich einfacher als in Assembler. Außerdem erspart sich der Programmierer eine Menge Arbeit, wenn er Dateizugriffe mit den Möglichkeiten der C Run-Time-Bibliothek realisiert. Für diejenigen unter Ihnen, die einen Microsoft C-Compiler besitzen, sind die C-Funktionen in eine Assemblerdatei RDOVLY.ASM (*Listing 3*) übersetzt und die zugehörigen Funktionen der C Run-Time-Bibliothek als Assemblercode in SUPPORT.ASM (*Listing 4*) enthalten. Der Code ist kommentiert und sollte sich selbst erklären, ich gebe hier aber trotzdem noch einen kurzen Überblick über die Funktionen.

OVRMGR erhält die Kontrolle von $$OVLINIT und installiert ovly_int als neue Interruptroutine für den INT $$INTNO-Handler, der alte Vektor wird dabei aufbewahrt. In dieser Fassung der Programme wird der alte Interruptvektor nicht zurückgeschrieben, aber in Ihrer eigenen Anwendung sollten Sie diese Funktion noch dem Programmende zufügen. Der Vektor steht dafür in old_ovint zur Verfügung. Nach Abschluß der Initialisierung führt OVRMGR einen FAR-Sprung zu $$MAIN aus, der ursprünglichen Startadresse des Programms. Wird ein Overlay aufgerufen, erhält OVRMGR über ovly_int erneut die Kontrolle. Zunächst prüft OVRMGR, ob noch ein Overlay aktiv ist, gibt in diesem Fall eine Fehlernachricht aus und beendet sich mit dem Fehlercode 41H. Damit wird verhindert, daß ein Overlay sich selbst oder ein anderes Overlay anfordert. Ist der Overlay-Manager beim Aufruf nicht aktiv, wird die Overlay-Nummer und der Offset aus der ursprünglichen Codierung übernommen. Die Segmentadresse des Overlay-Bereiches wird aus dem Feld $$MPGSNBASE gelesen und die Routine read_overlay_section gerufen, um das Overlay zu laden. Tritt beim Lesen der Datei mit dem ausführbaren Code ein Fehler auf, beendet sich OVRMGR mit dem Fehlercode 42H, andernfalls wird das Overlay gerufen. Darauf werde ich weiter unten noch genauer eingehen.

Die Routine read_overlay_section ist gradlinig aufgebaut. Zuerst wird die Datei mit dem ausführbaren Code unter dem Namen, den das ursprüngliche Programm in der Variablen executable_name vorgibt, geöffnet. Diese Variable ent-

◄ *Listing 3: (Fortsetzung)*

```
;
;               /* fix up relocations in the loaded overlay */
;
;               mylseek(executable_handle,startpos + (long)eh.first_reloc,0);
        mov     ax,0
        push    ax
        mov     ax,WORD PTR [bp-160]
        sub     dx,dx
        add     ax,WORD PTR [bp-8]      ;startpos
        adc     dx,WORD PTR [bp-6]
        push    dx
        push    ax
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _mylseek
        add     sp,8
;               while (eh.num_relocs) {
$FC164:
        cmp     WORD PTR [bp-178],0
        jne     $JCC403
        jmp     $FB165
$JCC403:
;                reloc_chunk = min(RELOC_BUF_SIZE,eh.num_relocs);
        mov     ax,WORD PTR [bp-178]
        sub     ax,32
        sbb     cx,cx
        and     ax,cx
        add     ax,32
        mov     WORD PTR [bp-186],ax    ;reloc_chunk
;    myread(executable_handle,reloc_buf,reloc_chunk * sizeof(struct reloc_entry_));
        mov     ax,WORD PTR [bp-186]    ;reloc_chunk
        shl     ax,1
        shl     ax,1
        push    ax
        lea     ax,WORD PTR [bp-140]    ;reloc_buf
        push    ss
        push    ax
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myread
        add     sp,8
;                eh.num_relocs -= reloc_chunk;
        mov     ax,WORD PTR [bp-186]    ;reloc_chunk
        sub     WORD PTR [bp-178],ax
;                for (i = 0; i < reloc_chunk; i++) {
        mov     WORD PTR [bp-142],0     ;i
        jmp     $F166
$FC167:
        inc     WORD PTR [bp-142]       ;i
$F166:
        mov     ax,WORD PTR [bp-186]    ;reloc_chunk
        cmp     WORD PTR [bp-142],ax    ;i
        jl      $JCC482
        jmp     $FB168
$JCC482:
;                    reloc_ptr = reloc_buf + i;
        mov     si,WORD PTR [bp-142]    ;i
        shl     si,1
        shl     si,1
        lea     ax,[bp-140][si]
        mov     WORD PTR [bp-146],ax    ;reloc_ptr
        mov     WORD PTR [bp-144],ss
;                    target_ptr = MK_FP(pspaddr + 0x10 + reloc_ptr->segment,
        les     bx,DWORD PTR [bp-146]   ;reloc_ptr
        mov     ax,WORD PTR es:[bx+2]
RDOVLY_TEXT      ENDS
CONST       SEGMENT
$T20002 DW SEG _pspaddr
CONST       ENDS
RDOVLY_TEXT      SEGMENT
        ASSUME  CS: RDOVLY_TEXT
        mov     es,$T20002
        add     ax,es:_pspaddr
        add     ax,16
        sub     dx,dx
        mov     dx,ax
        sub     ax,ax
        les     bx,DWORD PTR [bp-146]   ;reloc_ptr
        or      ax,WORD PTR es:[bx]
        mov     WORD PTR [bp-154],ax    ;target_ptr
        mov     WORD PTR [bp-152],dx
;                                       reloc_ptr->offset);
;                    *target_ptr += pspaddr + 0x10;
        mov     es,$T20002
        mov     ax,es:_pspaddr
        add     ax,16
        les     bx,DWORD PTR [bp-154]   ;target_ptr
        add     WORD PTR es:[bx],ax
;                }
        jmp     $FC167
$FB168:
;           }
        jmp     $FC164
$FB165:
;           myclose(executable_handle);
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myclose
        add     sp,2
;           return 0;
        mov     ax,0
        jmp     $EX139
;        } else {
        jmp     $I169
$I162:
;               nextpos = startpos + sectionsize;
        mov     ax,WORD PTR [bp-8]      ;startpos
        mov     dx,WORD PTR [bp-6]
        add     ax,WORD PTR [bp-4]      ;sectionsize
        adc     dx,WORD PTR [bp-2]
        mov     WORD PTR [bp-12],ax     ;nextpos
        mov     WORD PTR [bp-10],dx
;           /* round up to 512-byte bound */
;           nextpos = (nextpos + 511L) & ~511L;
        mov     ax,WORD PTR [bp-12]     ;nextpos
        mov     dx,WORD PTR [bp-10]
        add     ax,511
        adc     dx,0
        and     ax,-512
        and     dx,-1
        mov     WORD PTR [bp-12],ax     ;nextpos
        mov     WORD PTR [bp-10],dx
;               if (nextpos >= filesize) {
        mov     ax,WORD PTR [bp-150]    ;filesize
        mov     dx,WORD PTR [bp-148]
        cmp     WORD PTR [bp-10],dx
        jge     $JCC646
        jmp     $I170
$JCC646:
        jle     $JCC651
        jmp     $L20003
$JCC651:
        cmp     WORD PTR [bp-12],ax     ;nextpos
        jae     $JCC659
```

▶ *Listing 3: (Ende)*

```
        jmp     $I170
$JCC659:
$L20003:
;                myclose(executable_handle);
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myclose
        add     sp,2
;                  return 1;
        mov     ax,1
        jmp     $EX139
;           }
;               mylseek(executable_handle,nextpos,0);
$I170:
        mov     ax,0
        push    ax
        push    WORD PTR [bp-10]
        push    WORD PTR [bp-12]        ;nextpos
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _mylseek
        add     sp,8
;        }
$I169:
;    }
        jmp     $FC153
$FB154:
;    myclose(executable_handle);
        push    WORD PTR [bp-156]       ;executable_handle
        call    FAR PTR _myclose
        add     sp,2
;    return 1;
        mov     ax,1
        jmp     $EX139
;}
$EX139:
        pop     si
        pop     di
        mov     sp,bp
        pop     bp
        ret
_read_overlay_section   ENDP
        nop
RDOVLY_TEXT     ENDS
END
```

hält den vollständigen Pfadnamen der Datei als ASCIIZ-String, also in C-kompatibler Form. Der Pfadname ist von einem C-Programm einfach zu erhalten, da er sich direkt hinter dem DOS-Environment befindet, das jeder Prozeß erhält.

Die meisten Sprachen bieten Felder, die, ähnlich argv[] in C, die Übernahme von Argumenten ermöglichen. Ich will hier nicht näher darauf eingehen, wie man executable_name erzeugen kann. Diese Variable muß aber von dem Programm gesetzt werden, das die Overlay-Strukturen benutzt. In unserem Beispiel ruft das Hauptprogramm _main eine Prozedur fill_exe_name, die wiederum executable_name initialisiert (s. OVRMGR.ASM). Beim Austesten mit SYMDEB ist übrigens Vorsicht geboten, denn, wie sich bei meinen eigenen Versuchen herausgestellt hat, verschwindet unter SYMDEB die Pfadangabe, die sich eigentlich vor dem Dateinamen aus dem Environment befinden sollte. Wird OVASM.EXE direkt von der DOS-Ebene gestartet, ist der Pfad aber vollständig im Environment enthalten.

Wenn die durch executable_name bezeichnete Datei geöffnet ist, bestimmt read_overlay_section die Größe der Datei. Dazu wird die Funktion lseek und nicht, wie man es eigentlich erwarten könnte, die in C vorhandene Funktion tell, benutzt. Der Grund dafür ist, daß ich die Aufrufe von Funktionen aus der C Run-Time-Bibliothek einschränken wollte, denn den in C entwickelten und in Assembler übersetzten Programme muß ich diese speziellen Funktionen sowieso nochmal als Assemblerroutinen zufügen.

Danach wird der Zeiger auf die Position innerhalb der Datei wieder auf deren Anfang gestellt und read_overlay_section beginnt eine Endlosschleife, um das angeforderte Overlay zu suchen. Falls es während dieses Suchvorgangs irgendwann zu einer Fehlersituation kommt, wird über die übliche Ausgaberoutine errputs eine Fehlermeldung ausgegeben, die Datei geschlossen und ein Fehlercode ungleich 0 (hier speziell 1) zurückgegeben. Möglich sind Lesefehler, nicht interpretierbare Informationen innerhalb der Datei oder das Fehlen des angeforderten Overlays. Das die mit executable_name bezeichnete Datei vorhanden ist, ist hierbei allerdings Voraussetzung. Kann auf die Datei nicht zugegriffen werden, weil sie z.B. auf einer Diskette steht, die aus dem Laufwerk genommen wurde, erkennt die Routine das beim Lesen des EXE-Kopfes und gibt einen Fehler zurück.

Findet read_overlay_section das Overlay, wird das Segment gelesen und in den Bereich geladen, auf den der Parameter ovrea zeigt. Danach wird der Code verschoben, denn jede Referenz darauf muß mit der Ladeadresse des Programms übereinstimmen. Das ist die gleiche Verschiebung, die COMMAND.COM beim ersten Laden des Programms ausführt, wenn auch nur für den Ausgangsbereich. Der Overlay-Manager bestimmt dann die exakte Ladeadresse des geladenen Overlays. Damit die Ladeadresse des Programms mit den Referenzen innerhalb des geladenen Overlays übereinstimmen, setzt $$OVLINIT die Variable pspaddr für die Verschiebung. Die Ladeadresse des Programms ist grundsätzlich die Adresse von PSP+10H, da der PSP 100H Byte belegt.

Um das Overlay verschieben zu können, nutzt read_overlay_section die Variable pspaddr und die Verschiebetabelle aus dem Dateikopf. Dafür wird die Schleife while(eh.num_relocs) solange durchlaufen, bis alle Einträge der Verschiebetabelle verarbeitet sind. Die jeweiligen Adressen erhält man durch Addition der Segmentadressen aus der Tabelle und der Ladeadresse des Programms (pspaddr+0x10). Der so berechnete FAR-Zeiger beinhaltet also die Zieladresse innerhalb des verschobenen Overlay-Segments und Referenzen auf Code des Segments können damit angepaßt werden. Prinzipiell müssen alle Referenzen auf Segmente (relativ zu 0) in aktuelle, absolute Segmentreferenzen umgewandelt werden. In der Routine read_overlay_section wird dafür, unter Benutzung des berechneten Zeigers, die Ladeadresse des Programms (pspaddr+0x10) auf die Zieladresse addiert. Einer der Gründe, weshalb ich diese Routine in C entwickelt habe, war übrigens die Pufferung von Einträgen der Verschiebetabelle.

Ist die Verschiebung beendet, wird die Datei mit dem ausführbaren Code geschlossen und an OVRMGR.ASM der Returncode 0 zurückgegeben. Trat beim Laden ein Fehler auf, wird, wie ich oben bereits erwähnt habe, von ovly_int eine Fehlermeldung ausgegeben und die Verarbeitung mit dem Returncode 1 abgebrochen. Nach dem erfolgreichen Laden wird die Adresse der Marke return_from_ov auf den Stack gepusht und damit die Rückkehr aus der Overlay für ein RETF vorbereitet.

Für den Sprung zu den Overlay-Routinen werden deren Adressen ebenfalls auf den Stack gepusht und ein RETF ausgeführt. Der Einsprung in die Routinen über einen FAR-Zeiger wäre zwar genauso einfach, aber mir liegt die direkte Verwendung des Stacks mehr (ist natürlich reine Ansichtssache). Wenn die Overlay-Routine abgearbeitet ist, erfolgt über eine RETF-Anweisung (wir verwenden ausschließlich FAR-Adressen) der Rücksprung zu return_from_ov, damit aufgeräumt und das Flag, das anzeigt, ob der Overlay-Manager aktiv ist, zurückgesetzt werden kann. Danach geht die Kontrolle wieder an das Hauptprogramm, auf die Anweisung direkt nach dem Aufruf der Overlay-Routine.

Die Datei SUPPORT.ASM enthält bestimmte Assemblerfunktionen, auf die aus RDOVLY.ASM zugegriffen wird und die entsprechende Funktionen der C Run-Time-Bibliothek ersetzen sollen: errputs, myopen, myclose, mylseek und myread. Die Datei OVRMGR.ASM beinhaltet die Initialisierung und Routinen zur Handhabung der Interrupts. Die Datei RDOVLY.ASM, die aus RDOVLY.C durch Übersetzung entstanden ist, vervollständigt diesen einfachen Overlay-Manager.

Testprogramme

Dem Overlay-Manager sind einige Testroutinen beigefügt, die sowohl in Assembler, als auch in C codiert sind. Die Dateien OV.C, P1.C und P2.C (*Listing 5*) sind als einfacher Test gedacht, entsprechende Assemblerroutinen sind in OVASM.ASM, P1ASM.ASM und P2ASM.ASM (*Listing 6*) enthalten. Wenn sie Microsoft C Version 5.1 und MASM Version 5.1 verwenden, können Sie den gesamten Quellcode mit den Dateien OV und OVASM (*Listing 7*) und MAKE übersetzen. Natürlich müssen MASM und LINK im DOS-Pfad (Path-Kommando) bekannt sein und Ihre Version von LINK muß Overlays unterstützen. Der Quellcode der Datei RDOVLY.C, aus dem der Assemblercode in RDOVLY.ASM entsteht, kann durch das Programm SMERGE (*Listing 8*) weiter bearbeitet werden. SMERGE kopiert den C-Quellcode in die /Fa Assemblerausgabe von Microsoft C und entfernt die störende Definition EXTRN _acrtused:ABS, die der Compiler generiert. Sie können SMERGE.C auch leicht an andere C-Compiler anpassen.

Einschränkungen

Der hier beschriebene Overlay-Manager stellt nur ein Rudiment einer vollständigen Overlay-Verwaltung dar. Unter anderem kann die Fehlererkennung und -behandlung verbessert werden, z.B. ist hier nicht abgefangen, daß eine Diskette mit dem ausführbaren Code während des Programmlaufs aus dem Laufwerk genommen wird. Sicherlich benutzen die meisten PC-Anwender Festplatten, um ihre Programme, auch die mit Overlay-Strukturen, abzuspeichern, aber eine besondere Fehlerbehandlung der Overlay-Verwaltung würde die Sicherheit des gesamten Systems deutlich erhöhen. Zum Beispiel benutzt der Microsoft Overlay-Manager entweder die Diskette oder einen neuen Pfadnamen für die .EXE-Datei. Eine wichtigere Einschränkung ist die Zulassung von nur einem aktiven Overlay zu einem bestimmten Zeitpunkt. Das ist durch die Art und Weise, wie Informationen im Overlay-Manager abgelegt und zurückgegeben werden, bedingt. Es ist sicher nicht allzu schwer, OVRMGR.ASM so zu verändern, daß eine beliebige Anzahl von Overlays gleichzeitig unterstützt werden.

Der Overlay-Manager von Microsoft C erlaubt bis zu 128 verschachtelte Overlay-Aufrufe. Vielleicht ist das für Sie eine Anregung, unsere Version weiter zu entwickeln. In der Dokumentation zu MASM wird darauf hingewiesen, daß CodeView jetzt den Overlay-Manager unterstützt und es durchaus notwendig werden kann, große Anwendungen durch Overlays zu strukturieren, damit sie unter CodeView lauffähig sind. Tatsächlich existieren zwischen dem Microsoft Overlay-Manager und CodeView (und DEBUG)

```
                .model  LARGE,C
                .code

                public  errputs
errputs         proc    far C uses ax bx cx di ds es, string:ptr
                les     di,string                   ;es:di -> string
                push    di                          ;save original offset
                mov     al,0                        ;terminator byte
                mov     cx,0100h                    ;length to examine for null
                repne   scasb                       ;look for it
                pop     ax                          ;ax = orig pointer
                sub     di,ax
                dec     di                          ;di = length
                mov     cx,di
                mov     bx,2                        ;stderr
                mov     dx,ax                       ;buffer address
                mov     ah,40h                      ;write to file
                int     21h
                ret
errputs         endp

mylseek         proc    far C uses bx cx, handle:WORD, ofs:DWORD, whence:WORD
                public  mylseek
                mov     ax,whence
                mov     ah,42h
                mov     bx,handle
                push    es
                les     dx,ofs
                mov     cx,es
                pop     es
                int     21h                         ;result left in dx:ax, nicely
                jnc     lseekdone
                mov     ax,-1
lseekdone:      ret
mylseek         endp

myread          proc    far C uses bx ds dx, handle:WORD, buf:PTR, len:WORD
                public  myread
                mov     ah,3fH
                mov     bx,handle
                mov     cx,len
                lds     dx,buf
                int     21h                         ;result nicely in ax for return
                jnc     readdone
                mov     ax,-1
readdone:       ret
myread          endp

myopen          proc    far C uses ds dx, fname:PTR, mode:WORD
                public  myopen
                mov     ax,mode
                mov     ah,3dH
                lds     dx,fname
                int     21h                         ;result nicely in ax for return
                jnc     opendone
                mov     ax,-1
opendone:       ret
myopen          endp

myclose         proc    far C uses bx, handle:WORD
                public  myclose
                mov     ah,3eH
                mov     bx,handle
                int     21h                         ;result nicely in ax for return
                ret
myclose         endp


                end
```

◂ *Listing 4: SUPPORT.ASM*

Die Programmierung von Assemblerroutinen in C

Die Programmierung in Assembler bietet einige offensichtliche Vorteile: maschinennahe Kontrolle, hohe Geschwindigkeiten und geringen Programmumfang. Und das in einem Maße, wie es mit einer Hochsprache nicht erreicht werden kann. Die Codierung von Assemblerroutinen kann andererseits auch außerordentlich mühsam sein. Die Umsetzung von Algorithmen, insbesondere wenn es sich um die Verarbeitung von Feldern, der Realisierung von Schleifen mit komplizierten Abbruchbedingungen oder Dateizugriffen handelt, ist hochgradig anfällig für Programmierfehler, selbst wenn der Entwickler schon einige Erfahrung in Assemblerprogrammierung gesammelt hat. Aus diesen Gründen habe ich mich dafür entschieden, die Initialisierung und die Umgebung für die Overlay-Interrupts in MASM und das Hauptprogramm in Microsoft C zu codieren. Daraus ist die Funktion read_overlay_section in der Datei RDOVLY.C entstanden.

Da ich aber auch einen Overlay-Manager für die Entwickler in Assembler darstellen wollte, habe ich die C-Routinen teilweise in Assembler umgeschrieben, bis ich darauf gekommen bin, daß die Option /Fa von Microsoft C, die MASM-kompatiblen Assemblercode in eine Datei generiert, mir bei der Umsetzung helfen kann.

Als erstes habe ich die Routinen, die ich aus der C Run-Time-Bibliothek benötigte, neu in Assembler entwickelt. Das war mit einiger vorausschauender Überlegung nicht allzu schwer. Da sich viele systemnahe Funktionen sehr einfach mit Hilfe des DOS-Interrupts (INT 21H) realisieren lassen, konnte ich z.B. die DOS-Funktion 42H für meine Routine mylseek zur Bestimmung der aktuellen Dateiposition benutzen und auf die entsprechende C-Funktion verzichten. Meine eigene lseek-Funktion wird an verschiedenen Stellen benötigt und die korrekte Version liefert dasselbe Ergebnis wie die C-Funktion tell. Um die Übersetzung zu vereinfachen, habe ich Dateizugriffe auf die Ebene von DOS-Funktionen beschränkt. Die Routinen in SUPPORT.ASM ersetzen somit die Routinen der C Run-Time-Bibliothek und werden durch eine zusätzliche Funktion errputs zur Ausgabe auf stderr ergänzt.

In den Make-Dateien zu den Beispielen sind alle Steuerparameter für den Microsoft C-Compiler CL angegeben:

`/c` der Quellcode wird nur compiliert, nicht gebunden.

`/Fa` Generierung des Assemblercodes.

`/Fonul` die .OBJ-Dateien werden auf das Device NUL geleitet, also gelöscht.

`/AL` Benutzung des LARGE-Modells, d.h. es werden FAR-Adressen für Daten und Code erzeugt, damit LINK den Overlay-Code binden kann.

`/Gs` Aufrufe von _chkstk, also dem Test auf Stack-Überlauf, werden unterdrückt. Diese (versteckten) Run-Time-Routinen passen nicht mit meinem Assemblercode zusammen.

`/Zl` das Binden mit Modulen aus der Bibliothek in die .OBJ-Datei wird ausgeschlossen. Ich erzeuge die .ASM-Dateien zwar direkt, aber um völlig Sicher zu gehen, gebe ich diesen Parameter mit an

`/Od` der compilierte Code wird nicht optimiert und ist damit leichter zu übernehmen.

Da der vom Compiler erzeugte Assemblercode Referenzen auf Zeilen im ursprünglichen C-Quellcode enthält, habe ich das Programm SMERGE.C beigefügt, mit dem die Kommentare

`; Line <n>`

automatisch durch die originalen C-Anweisungen ersetzt werden. Das verdeutlicht den Zweck des zugehörigen Assemblercodes. Darüber hinaus entfernt SMERGE die Datendeklaration für eine externe Variable _acrtused, die CL zusätzlich generiert und deren Bedeutung mir bis heute nicht klar ist. Auf diese Weise erhält man mit CL und SMERGE eine Datei mit sauberem, gut kommentiertem Assemblercode.

Den generierten Code habe ich nicht weiter optimiert, obwohl sicherlich die Leistungsfähigkeit gesteigert und mnemonische (sprechende) Bezeichnungen eingefügt werden könnten. Aber ich finde den von CL erzeugten Assemblercode durchaus verständlich und habe darum auf weitere Änderungen darin verzichtet.

Diese Prozedur zur Übersetzung von C in Assembler habe ich mit Erfolg an RDOVLY.C ausprobiert und konnte auf diese Weise an einem einzigen Nachmittag einen funktionierenden Overlay-Manager entwickeln, der sogar in C schon einigermaßen leistungsfähig war. Nach ein paar weiteren Stunden war die Version mit reinem Assemblercode lauffähig. Hätte ich direkt mit der Entwicklung in Assembler begonnen, wären Programmänderungen, z.B. an der Fehlerbehandlung zum Lesen der Overlay-Segmente oder der Pufferung von verschobenen Datenblöcken wesentlich schwieriger ausgefallen. Aber in C sind solche Programmänderungen einfach zu handhaben und der generierte Assemblercode ist einigermaßen leistungsfähig, da C an sich schon für maschinennahe Programmierung geeignet ist. Der Entwurf von Assemblerroutinen in C ist also durchaus sinnvoll. Probieren Sie es aus, es wird Ihnen gefallen.

Dan Mick

OV.C

```c
#include <stdio.h>
#include <string.h>

void p1(void);
void p2(void);

extern char executable_name;

void main(int argc, char **argv)
{
    int i;

    strcpy(&executable_name,argv[0]);
    argc = argc;

    printf("This is main\n");
    for (i = 0; i < 1000; i++) {
        p1();
        p2();
    }
}
```

P1.C

```c
#include <stdio.h>
void p1()
{
   printf("In p1, whose address is %Fp\n",p1);
   printf("\t 3 * 3 + 9 = %d\n",3*3+9);
   printf("\t 4 * 4 / 5 = %d\n",4*4/5);
}
```

P2.C

```c
#include <stdio.h>
void p2()
{
   printf("In p2, whose address is %Fp\n",p2);
   printf("\t 10 %% 3 = %d\n",10 % 3);
}
```

einige interne Verbindungen, so daß über besondere Aufrufe auf neue Symbole zugegriffen werden kann. Unser Overlay-Manager unterstützt sicherlich keine undokumentierten Funktionen, aber es hat sich herausgestellt, daß die Fehlersuche mit einem Debugger trotzdem möglich ist. Und zwar kann ein symbolischer Debugger Symboltabellen lesen, nachdem das Programm geladen ist, die .MAP-Datei sollte nur keine Symbole enthalten, die sich auf Overlay-Routinen beziehen. Da ohnehin keine Datensegmente Overlays zugeordnet werden können, ist das Problem nicht allzu gravierend. Meistens beschränkt sich die Fehlersuche ohnehin auf Datenlabels. Trotz aller Schwierigkeiten ist das Debuggen von Programmen mit Overlay-Strukturen durchaus interessant. Die Benutzung von Overlay-Verwaltungen bietet eine Lösung des Problems von begrenztem Speicherplatz.

In diesem Artikel habe ich einige Programmiertechniken aufgezeigt, mit beschränktem Speicherplatz umzugehen. Dabei bin ich besonders auf die Lösung von Microsoft eingegangen und habe versucht zu erklären, wie ein MASM-Programmierer mit dieser Art von FAR-Adressierung arbeiten kann. Wenn Ihnen der hier dargestellte Overlay-Manager nicht allzu brauchbar erscheint, können Sie ja meine Anregungen aufnehmen und die Programme weiterentwickeln, damit Sie ein passendes Werkzeug erhalten. Es lohnt sich auf jeden Fall, Overlay-Techniken genauer zu untersuchen.
Dan Mick

Dan Mick arbeitet als Elektroingenieur bei Definicon Systems in Newbury Park, Kalifornien.

OVASM.ASM

```
                extrn   _errputs:far, p1:far, p2:far
                extrn   _executable_name:byte
_text           segment public 'CODE'
_text           ends
_data           segment public 'DATA'
_data           ends

                public  fill_exe_name,ENVSEG,main_msg,call_loop,envstr_loop
                public  fname_loop

ENVSEG          equu    2CH                             ;loc'n of environment pointer
DGROUP          group   _data,_stack
_text           segment
                assume  cs:_text,ds:DGROUP,ss:DGROUP
_main           proc    far
                mov     ax,DGROUP
                mov     ds,ax
                call    far ptr fill_exe_name
                lea     ax,main_msg
                push    ds
                push    ax
                call    far ptr _errputs
                add     sp,4
                mov     cx,1000
call_loop:
                call    far ptr p1
                call    far ptr p2
                loop    call_loop
                mov     ax,4c00h
                int     21h
_main           endp

fill_exe_name   proc    far
                push    es
                push    di
                push    ds
                push    si
                push    ax

                mov     ax,es
                mov     ds,ax
                mov     ax,DGROUP:ENVSEG                ;point at env
                mov     ds,ax
                mov     si,0
                mov     ax,seg _executable_name         ;point at dest string
                mov     es,ax
                lea     di,DGROUP:_executable_name
envstr_loop:
                lodsb                                   ;get first env
                cmp     al,0                            ;end of string?
                jnz     envstr_loop                     ;no, keep looking
                cmp     byte ptr ds:[si],0              ;end of env?
                jnz     envstr_loop                     ;no, get another string
                add     si,3                            ;yes: go past, 2 more
fname_loop:     movsb                                   ;move a byte
                cmp     byte ptr ds:[si],0              ;end of name?
                jnz     fname_loop                      ;no, keep it up
                mov     byte ptr es:[di],0              ;store zero terminator

                pop     ax
                pop     si
                pop     ds
                pop     di
                pop     es
                ret
fill_exe_name   endp

_text           ends


_data           segment
main_msg        db      'This is main...',13,10,0
_data           ends


_stack          segment stack 'STACK'
                dw      200 dup (?)
_stack          ends

                end     _main
```

P1ASM.ASM

```
                extrn   _errputs:far
                .model  LARGE
                .data
p1_msg          db      'This is p1...',13,10,0

                .code
p1              proc    far
                public  p1
                lea     ax,p1_msg
                push    ds
                push    ax
                call    _errputs
                add     sp,4
                ret
p1              endp

                end
```

P2ASM.ASM

```
                extrn   _errputs:far
                .model  LARGE
                .data
p2_msg          db      'This is p2...',13,10,0

                .code
p2              proc    far
                public  p2
                lea     ax,p2_msg
                push    ds
                push    ax
                call    _errputs
                add     sp,4
                ret
p2              endp

                end
```

◄◄ *Listing 5: OV.C, P1.C und P2.C.*

◄ *Listing 6: OVASM.ASM, P1ASM.ASM und P2ASM.ASM.*

► *Listing 7: Die Make-Dateien OV und OVASM.*

►► *Listing 8: SMERGE.C.*

OV

```
ov.obj: ov.c
  cl /c /AL /Fm ov.c

p1.obj: p1.c
  cl /c /AL /Fm p1.c

p2.obj: p2.c
  cl /c /AL /Fm p2.c

ovrmgr.obj: ovrmgr.asm
  masm /MX ovrmgr.asm;

rdovly.asm : rdovly.c exehdr.h
# /c compile only
# /Fa create .ASM output
# /Fonul don't create object (leave that for MASM)
# /Al large model
# /Gs no stack checking
# /Zl no default library search
# /Od no optimization (for code clarity)
  cl /c /Fa /Fonul /Od /AL /Gs /Zl rdovly.c
  smerge rdovly.asm rdovly.c rdovly.mrg
  del rdovly.asm
  ren rdovly.mrg rdovly.asm

rdovly.obj : rdovly.asm
  masm /mx rdovly;

support.obj : support.asm
  masm /mx support;

ov.exe : ov.obj support.obj rdovly.obj ovrmgr.obj p1.obj p2.obj ovrmgr.obj
  link /m/li ov+support+rdovly+(p1)+(p2)+ovrmgr,ov,ov.map;
```

OVASM

```
ovasm.obj: ovasm.asm
  masm /mx ovasm.asm;

plasm.obj: plasm.asm
  masm /mx plasm;

p2asm.obj: p2asm.asm
  masm /mx p2asm;

ovrmgr.obj: ovrmgr.asm
  masm /MX ovrmgr.asm;

rdovly.asm : rdovly.c exehdr.h
# /c compile only
# /Fa create .ASM output
# /Fonul don't create object (leave that for MASM)
# /Al large model
# /Gs no stack checking
# /Zl no default library search
# /Od no optimization (for code clarity)
  cl /c /Fa /Fonul /Od /AL /Gs /Zl rdovly.c
  smerge rdovly.asm rdovly.c rdovly.mrg
  del rdovly.asm
  ren rdovly.mrg rdovly.asm

rdovly.obj : rdovly.asm
  masm /mx rdovly;

support.obj : support.asm
  masm /mx support;

ovasm.exe : ovasm.obj support.obj rdovly.obj ovrmgr.obj plasm.obj p2asm.obj
            ovrmgr.obj
  link /m/li ovasm+support+rdovly+(plasm)+(p2asm)+ovrmgr,ovasm,ovasm.map;
```

```
/* smerge - merge source and assembly files */


#include <stdio.h>


main(argc,argv)
int argc;
char *argv[];
{
    FILE *csrc, *asmsrc, *mergefile;
    char buffer[128];
    long sline, line, totslines;


    if (argc < 4)
    {
        printf("usage: smerge asmsrc csrc mergefile\n");
        exit(1);
    }


    sline = 0;


    asmsrc = fopen(argv[1],"r");
    csrc = fopen(argv[2],"r");
    mergefile = fopen(argv[3],"w");


    setvbuf(csrc,NULL,_IOFBF,8192);
    setvbuf(asmsrc,NULL,_IOFBF,8192);
    setvbuf(mergefile,NULL,_IOFBF,8192);


    totslines = 0;
    while (fgets(buffer,128,csrc) != NULL)
        totslines++;
    fseek(csrc,0L,SEEK_SET);


    while ((fgets(buffer,128,asmsrc)) != NULL)
    {

        /* special hack for MSC -> assembly...kill the "__acrtused" extrn */
        if (strstr(buffer,"__acrtused") != NULL)
            continue;


        if (strncmp(buffer,"; Line",6) == 0)
        {
            sscanf(buffer,"; Line %ld",&line);
            if (line <= totslines)
            {
                while (sline < line)
                {
                    fgets(buffer+1,128,csrc);
                    fputs(buffer,mergefile);
                    ++sline;
                }
            }
        }
        else
            fputs(buffer,mergefile);
    }


    fclose(asmsrc);
    fclose(csrc);
    fclose(mergefile);

}
```

Das universelle Tool PLUS:

System-unabhängige Applikationen erstellen

Die Ziele heutiger Anwendungsprogrammierungen gehen in mehrere Richtungen. Zum ersten wird ein Interface erwünscht, das »benutzerfreundlich« ist. Dies bedeutet im allgemeinen, daß eine Oberfläche angeboten wird, die grafisch orientiert ist und eine Metapher aus dem alltäglichen Leben benutzt. Das zweite wichtige Ziel ist Portabilität.

Die Metapher »Benutzerfreundlichkeit« ist häufig der »Desktop«, der Schreibtisch. Auf diesem Schreibtisch können »Dinge« bewegt, entfernt, verändert und erzeugt werden. Zentraler Punkt für das einfache Arbeiten mit dem System ist die Verwendung eines Zeigegerätes, dessen verbreitetstes Form wohl die Maus ist. Mit der Maus werden auf dem Bildschirm Objekte ausgewählt, modifiziert oder über Menüs werden Operationen aktiviert. Die Metapher ist dabei den jeweiligen Anforderungen, also dem *richtigen Leben* anzupassen.

Das zweite Ziel ist Portabilität. Eine einmal erstellte Anwendung sollte auf möglichst vielen Umgebungen einsetzbar sein. Die Erweiterbarkeit und Veränderbarkeit einer Anwendung durch den Benutzer trägt wesentlich zu Akzeptanz in ihrem eigentlichen Einsatzbereich bei. Nicht zuletzt soll die Erstellung der Anwendung effektiv möglich sein und sich in eine gute Wartbarkeit fortsetzen.

Das universelle Tool PLUS bietet die Basis zum Erreichen all dieser Ziele.

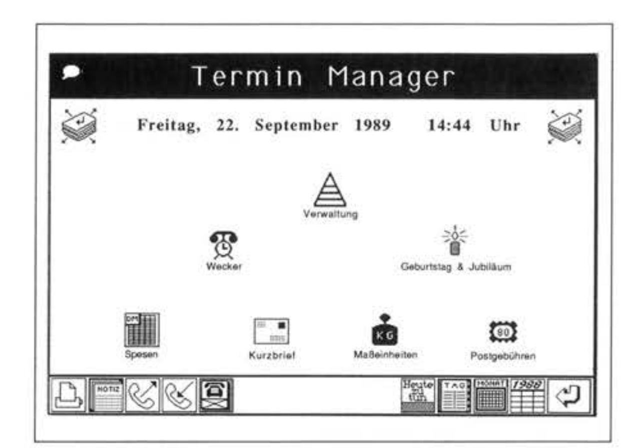
Allgemeines

PLUS ist eine Applikation, die einerseits die Funktion eines hochwertigen Editors übernimmt, in dem die zu erstellenden Anwendungen entwickelt und gewartet werden. Andererseits stellt PLUS die Laufzeitumgebung bereit, in der die Anwendungen ausgeführt werden. Beide Funktionen sind eng miteinander verbunden und liefern dem Entwickler ein homogenes Erscheinungsbild. Nicht zuletzt diese Eigenschaft führt zu einem sehr effektiven Arbeiten, das sich neben einem schnellen Entwickeln auch in qualitativ hohen und ansprechenden Ergebnissen ausdrückt.

Es soll im folgenden ein kurzer Überblick über das Werkzeugt PLUS gegeben werden, wobei vor allem die Grundkomponenten und Ideen klargemacht werden sollen.

Stacks, Karten und Hintergründe

PLUS dient dem Bearbeiten von passiven, persistenten Objekten, die mit Stacks bezeichnet werden. Diese Stacks werden hierzu auf normale Dateien im Filesystem abgebildet. Die Stacks sind ihrerseits weiter dadurch strukturiert, daß sie sich aus mehreren Karten zusammensetzen. Mehrere Karten können zu einem gemeinsamen Hintergrund gehören, von dem sie gewisse Eigenschaften erben. Karten können zusammen mit den Informationen ihres Hintergrunds in einem Fenster des Rechners dargestellt werden, wobei die Größe der Karten frei eingestellt wer-



▸ *Terminmanager*

▸▸ *Tagesplan*

den kann. Es ist nun die hauptsächliche Aufgabe des Entwicklers, die Karten zu gestalten und dafür zu sorgen, daß Karten angezeigt werden. Für die Gestaltung stehen ihm zahlreiche Möglichkeiten zur Verfügung. Zum einen können Karten und Hintergründe mit Grafiken versehen werden, wofür ein komfortabler Bitmap-Editor in PLUS integriert ist, eigentlich ein vollständiges Malprogramm. Man kann sich die Karte in der Art einer Overhead-Folie über dem Hintergrund liegend vorstellen, die somit das Aussehen der Karte in den Teilen verändern kann, die nicht bei allen Karten dieses Hintergrunds gleich sind.

Eine weitere und wohl die wichtigste Möglichkeit zur Gestaltung der Karten und Hintergründe ist das Einbringen von weiteren Objekten, die frei über den Grafiken arrangiert werden können und an die Funktionalität gebunden werden kann. Es steht ein Vielzahl von Objektarten zur Verfügung, die später beschrieben werden.

Scripts und Vererbung

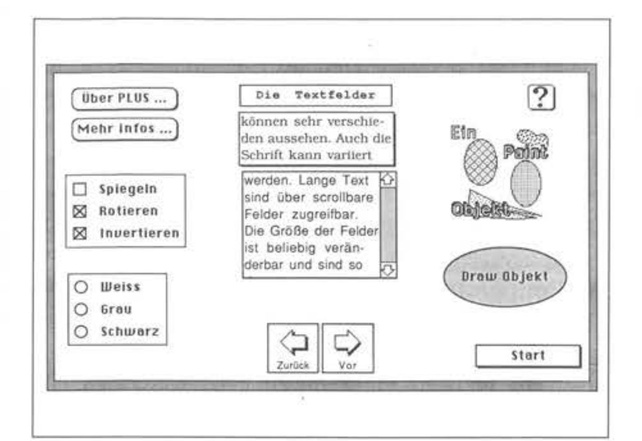
Von ganz zentraler Bedeutung für die Flexibilität von PLUS ist die interpretierte Programmiersprache PPL, ein Superset der Sprache HyperTalk von Apple. Sie lehnt sich an das Englische an und ist deshalb leicht erlernbar und lesbar. Durch sie kann fast jede Eigenschaft von PLUS verändert werden. Dies betrifft zum einen die Objekte wie Stack, Karte und Hintergrund sowie die Objekte auf den Karten und globale Eigenschaften von PLUS. Weiterhin kann auf ein großes Angebot von Funktionen zurückgegriffen werden. Die Möglichkeiten von PPL können direkt durch Eingabe in ein Fenster abgerufen werden oder sie werden durch selbstgeschriebene Routinen, Handler genannt, erweitert. Hierzu steht dem Programmierer ein Satz von Kontrollstrukturen zu Verfügung, wie sie aus höheren Programmiersprachen bekannt sind. Mehrere Handler werden zusammengefaßt zu einem Script und ein solches Script kann dann an ein Objekt gebunden werden. Zum Erstellen der Scripts wird ein intelligenter Editor bereitgestellt, der neben einer integrierten, intelligenten Hilfsfunktion, die zum halbautomatischen Programmieren benutzt werden kann, das Formatieren der Scripts übernimmt.

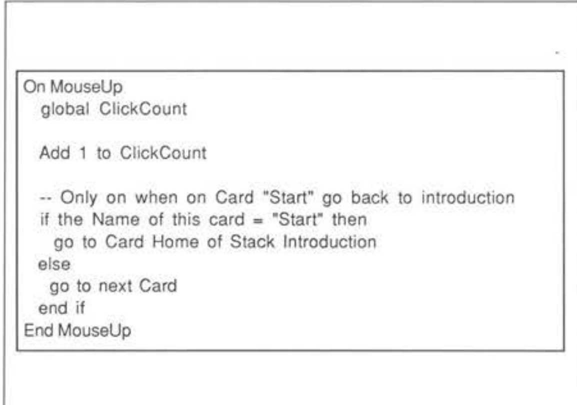
Die Scripts von Karte, Hintergrund und Stack sind hierachisch organisiert, d.h., eine Karte erbt die Handler ihres Hintergrunds bzw. Stacks. Dies ermöglicht es, gemeinsame Funktionalität für Karten und Hintergründe zu definieren, was zu wohlstrukturierten Programmen führt.

Die Applikationen werden in einer ereignisorientierten Weise aufgebaut. Dazu werden von PLUS Systemnachrichten erzeugt, die über das Eintreten von Ereignissen informieren. Diese Nachrichten können dann von Handlern mit den entsprechenden Namen abgefangen und bearbeitet werden. Einige Bespiele für Systemnachrichten sind MOUSEDOWN, die das Drücken der Maus signalisiert oder OPENCARD, die angibt, daß eine Karte neu angezeigt wird. Die Systemnachrichten sind die Auslöser für Aktionen.

Tasten, Felder und andere Objekte

Es wurde bereits erwähnt, daß zur Gestaltung von Karten Objekte auf die Karten gelegt werden können. Eine Art von Objekten ist die Taste. Eine Taste kennzeichnet einen einfachen geometrischen Bereich auf der Karte, in dem die Taste in einer einstellbaren Weise angezeigt wird. Die Größe und Position der Taste sind interaktiv beliebig veränderbar. Jede Taste hat einen Namen, eine ID und eine Nummer, über die die Taste identifiziert werden kann. Diese Identifizierung wird dann in der Programmiersprache benutzt, um die Eigenschaften der Taste zu erfragen bzw. zu ändern. Auch eine Taste kann ein Script besitzen, das dann z.B. die Systemnachricht MOUSEDOWN abfangen und für diese Taste bearbeiten kann. Die Eigenschaften der Taste lassen sich sowohl über die Programmiersprache als auch über einen Dialog direkt verändern. Tasten werden in der Anwendung vor allem dort benutzt, wo durch einen Mausklick eine Aktion ausgelöst werden soll. Icons, das sind kleine Symbole, die in der Taste angezeigt werden können, erlauben das bildhafte Deutlichmachen der mit dieser Taste verbundenen Aktivität.





```
On MouseUp
  global ClickCount

  Add 1 to ClickCount

  -- Only on when on Card "Start" go back to introduction
  if the Name of this card = "Start" then
    go to Card Home of Stack Introduction
  else
    go to next Card
  end if
End MouseUp
```

◄◄ *Beispiel für ein Script, das auf MouseUp reagiert.*

◄ *Einige Objekte*

Die anderen Arten von Objekten haben mit den Tasten vieles gemeinsam, wie etwa das Vorhandensein von Scripts, einer Position und Größe, einen Namen, eine ID und anderes mehr. Darüber hinaus können die Objekte Daten enthalten. Ein Beispiel ist das Textfeld. Es erlaubt die Eingabe und das Editieren von Texten. Die Texte können formatiert, in Zeichensatz, Zeichengröße und Zeichenart verändert und über die Programmiersprache manipuliert werden. Eine besondere Eigenschaft von Textfeldern erlauben ihren Einsatz für den Aufbau von Eingabemasken. Denn Textfelder im Hintergrund speichern ihre Texte in den Karten, d.h., es läßt sich mit dem Textfeld die Struktur eines Datensatzes definieren, und jede Karte enthält dann einen Datensatz mit den verschiedenen Informationen. Eine Variante von den Textfeldern, die DBFelder, sind gerade in Hinblick auf diesen Anwendungsbereich konzipiert worden. Mit ihnen kann ein Format vorgegeben werden, gegen das die Eingabe geprüft wird, z.B für Zahleneingaben.

Speziell für grafische Effekte und Animationen sind die Paint-Objekte geeignet. Sie können genau wie die Grafik von Karte und Hintergrund gestaltet werden, sind darüber hinaus aber genau wie Tasten frei auf der Karte plazierbar. Neben den Paint-Objekten erlauben die Draw-Objekte das Arbeiten mit einfachen geometrischen Figuren.

Navigation

Ist das Aussehen des Stack durch seine Karten definiert, besteht nun die Aufgabe des Programmierers darin, die erwünschte Funktionalität zu implementieren. Hierbei steht ihm zum einen die aus den konventionellen Programmiersprachen bekannten Möglichkeiten zur Verfügung, wie etwa die Definition von Prozeduren und Funktionen, das Arbeiten auf globalen und lokalen Variablen, Ausgabe von Texten und ähnliches mehr. Ferner kann kann er die Fähigkeit, auf die Objekte zuzugreifen und zu manipulieren, für sein Ziele einsetzen. Der wohl aber innovativste Aspekt der Programmierung ist die Navigation.

Durch Menüs oder PPL kann zwischen den einzelnen Karten hin und herbewegt werden. Dies kann für die verschiedensten Zwecke benutzt werden. Die einfachste Verwendung findet diese Möglichkeit in dem »Blättern« durch eine Stack. Damit ist gemeint, daß der Anwender sich durch Betätigen von Tasten durch den Stack bewegt, sich also verschiedene Karten ansieht. Dies führt bei gut aufgebauten Stacks zu einer sehr intuitiven Weise, Wissen darzustellen und abzurufen. Neben dieser einfachen Möglichkeit, werden die Navigationsfähigkeiten etwa für Animationseffekte, Wechseln von Eingabe- oder Ausgabemasken und anderes mehr benutzt.

Zur einfacheren Programmierung dieses Navigationsapektes ist in das System interaktives »Linking« realisiert worden. Hiermit kann auch ein Benutzer, der mit der Programmiersprache nicht vertraut ist, Stacks erzeugen und mit Tasten versehen, die dann dafür sorgen, daß zu verschiedenen Karten verzweigt wird. Dies wird vor allem bei der Gestaltung von HyperMedia-Systemen angewandt, bei denen das Aktivieren eines Begriffs oder eines Unterthemas zum Anzeigen einer anderen Karte führen soll.

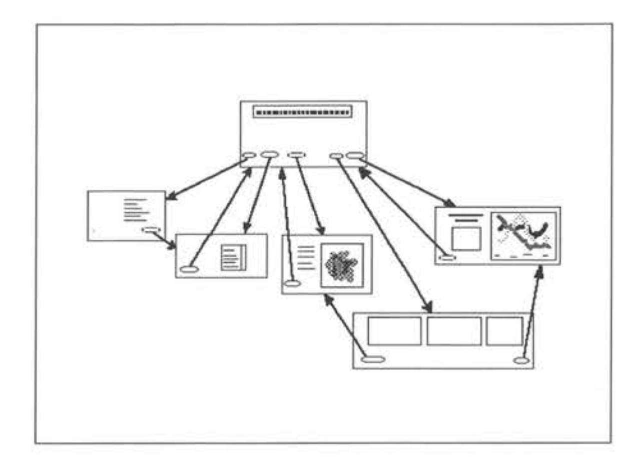
Volltextsuche

Für Anwendungen, die sich mit dem Information Retrieval beschäftigen, wurde eine sehr effektive Volltext-Suchfunktion eingebaut. Mit ihr ist es möglich, in allen Texten, oder auch in ausgewählten Texten eines Stacks nach Teiltexten zu suchen. Aufgrund der hohen Geschwindigkeit, mit der diese Suche möglich ist, können einfachere Datenbankanwendungen auf dieses Feature zurückgreifen.

Grafik

Der auf dem ersten Blick augenfälligste Teil der Stacks in PLUS ist die Grafik. Ein umfangreicher Satz von interaktiven Hilfsmitteln wird angeboten, um eine einfache grafische Gestaltung zu erlauben. Hierzu gehört neben dem Erstellen einfacher geometrischer Grundfiguren wie Linien,

▶ *Beispiel für »verlinkte« Karten.*



Rechtecke, Ovale, Polygone und Texten das Manipulieren dieser Strukturen. So können beliebige Teile der Grafik gelöscht, kopiert, verschoben oder mit den verschiedensten Effekten, wie etwa Rotieren, verändert werden. Die Grafik ist dabei bitmaporientiert und unterstützt Farbe. Gerade der Einsatz von Farbe ist für ein benutzerfreundliches Erscheinungsbild von großer Bedeutung. Ein wohlüberlegter Einsatz der Grafik erleichtert es dem Anwender, sein bekanntes Umfeld auf die Anwendung zu übertragen und somit effektiver zu arbeiten.

Benutzerlevels und Runtime-Version

Wie bereits erwähnt, stellt PLUS zugleich ein Entwicklungssystem wie auch eine Laufzeitumgebung dar. Nun kann es für den Entwickler wichtig sein, daß der Anwender an der erstellten Applikation nicht in dem Umfang manipulieren kann, der dem Entwickler zusteht. Dazu wurden den Stacks und auch der laufenden Anwendung ein Benutzerlevel zugeordnet, daß die Zugriffsmöglichkeiten auf den Stack einschränkt. Dieser Level kann dann durch ein Paßwort geschützt werden. Die Einschränkungen für das System sind dabei gestaffelt und können immer an die aktuelle Situation angepaßt werden.

Da für das Ausführen einer mit PLUS erstellten Anwendung eine Laufzeitumgebung erforderlich ist, wird neben der Vollversion eine freie Runtime-Version angeboten, mit der dann die Anwendung ausgeführt, aber nicht interaktiv editiert werden kann. Dadurch sind der Erwerb und das Ausführen von PLUS-Anwendungen nicht an den Kauf des Entwicklungssystems gebunden, sondern die Anwendung kann als ein Stand-Alone-Programm aufgefaßt werden.

Erweiterbarkeit

Sollte die durch PLUS angebotene Funktionalität den Erfordernissen nicht genügen, so kann sie unter Nutzung des Software-Slot-Konzepts erweitert werden. Dies bedeutet z.B., daß der Programmierer in einer konventionellen Sprache wie Pascal oder C eine Funktion schreibt, diese kompiliert und den Code in das System integriert. Diese neue Funktion ist vollständig in die Sprache eingebunden, kann auf das System zugreifen und steht nun der Anwendung zur Verfügung. Für die Funktion stehen all die Möglichkeiten des Betriebsystems zur Verfügung. So könnten etwa auf Peripherie zugegriffen sowie Steuerungs- und Kontrollaufgaben bewältigt werden. Die Erweiterbarkeit geht sogar soweit, daß der Programmierer neue Objektarten erstellen kann, die in PLUS aufgenommen werden können. Denkbar sind etwa Objekte, die ein Spreadsheet oder eine Business-Grafik darstellen. Wichtig ist hierbei, daß alle Erweiterungen, seien es neue Funktionen, Kommandos oder Objekte, homogen in das System integriert werden.

Portabilität

PLUS ist ein portables Tool, das auf den Systemen MacOS, OS/2-Presentation Manager und Windows arbeiten kann. Die Stacks sind zwischen diesen Systemen frei austauschbar und unter allen manipulierbar. Dies eröffnet weitreichende Perspektiven für den Einsatz in gemischten Umgebungen. PLUS kann damit zum Austausch von Informationen zwischen den verschiedenen Systemen dienen und auf allen Systemen können die gleichen Anwendungen benutzt werden.

Anwendungen

Der Einsatzbereich von PLUS ist fast unbegrenzt. Es beginnt mit dem Erstellen kleiner Datenbanken, die beispielsweise auch grafische Informationen enthalten können. Die grafischen Fähigkeiten, zusammen mit den Möglichkeiten der Programmiersprache, machen es zu einer guten Basis für Präsentationen, die dann auch Animationen enthalten und selbständig ablaufen können. Konkret lassen sich Terminkalender, Notizsysteme, Telefonverzeichnisse, Adreßverwaltungen, Ersatzteilkataloge, Abfragesysteme, Spiele, Nachschlagewerke und anderes mehr erstellen.

Ein sehr interessanter Bereich ergibt sich zusammen mit den Software-Slots. PLUS kann damit zu einem Front-End-System ausgebaut werden, das etwa die Informationen einer externen Datenbank leicht zugreifbar macht.

Die Mächtigkeit des Systems und die leichte Modifizierbarkeit machen PLUS zu einer idealen Umgebung für Rapid Prototyping-Projekte. So könnten verschiedene Benutzerschnittstellen in minutenschnelle erzeugt und erprobt werden.

Udo Borkowski
Format Software GmbH, Brückenstr. 42
5000 Köln 50, Tel.: (0221) 35 10 77

NEU MICROSOFT WORD 5.0

Jetzt. MICROSOFT WORD 5.0 in deutsch ist da! Textverarbeitung, Textveredelung wie sie nur der Marktführer bieten kann. Neue Funktionen und Erweiterungen, die Ihre Texte und Dokumente auf Hochglanz bringen. Fortschritt, der fünffach überzeugt:

- Einfache und schnelle Einbindung von Grafiken, Bildern und Tabellen.
- Freie Positionierung von Text und Grafiken an jeder beliebigen Stelle.
- Layout-Kontrolle durch Ganzseiten-Ansicht vor dem Ausdrucken mit WYSIWYG von Text und Grafik.
- Mehrspaltige Texte, direkt am Bildschirm sichtbar.
- Deutscher Thesaurus.

MICROSOFT WORD 5.0 ist soeben in deutscher Version bei Ihrem Fachhändler eingetroffen. Informieren Sie sich detailliert über alle Vorteile. Bei ihm werden Sie feststellen, es gibt keine Alternative. Und wenn Sie schon mit WORD 4.0 arbeiten, dann gibt es nur eins: Updaten zum Superpreis!



Microsoft®
ZUKUNFT DER SOFTWARE

Coupon

Bitte senden Sie mir Informationsmaterial zu MICROSOFT WORD 5.0.

Ich nutze Software: ☐ privat ☐ beruflich/Branche ____________

Mein Rechner: ☐ MS-DOS ☐ MS OS/2

Bitte senden Sie den Coupon an: Microsoft Info-Service, Postfach 129, 8000 München 1

Absender nicht vergessen.

Mikroprozessor-Grundlagen:

Die Debugregister des Intel 386

In diesem Artikel besprechen wir die Möglichkeiten des 386 und erkunden, wie neuere Debugger diese unterstützen.

In der Vergangenheit...

Früher konnten Debugger, die auf dem PC liefen, nur Breakpunkte im Programm setzen, falls sich dieses im RAM befand. Breakpunkte, die den Datenzugriff überwachten, waren nicht möglich. Hilfsprogramme zur Datenüberwachung konnten verwendet werden. Diese verwendeten aber den Timerinterrupt, um periodisch die Daten zu prüfen und anzuzeigen. Ein Debugger überschrieb typischerweise das erste Byte des Befehls, was die Verwendung im schreibgeschützten Speicher (ROM) unmöglich machte. Setzten Sie einen Breakpunkt auf einen Befehl, so rettete der Debugger dieses Byte und schrieb an die Stelle einen 0CCH-Befehl (INT 3). Wenn der Mikroprozessor den 0CCH-Opcode ausführt, so erzeugt er einen Interrupt 3, und der Debugger, der den Interrupt 3 überwachte, wurde aufgerufen. Der Debugger zeigte die Breakpunkt-Anweisung und Register an und wartete auf ein Kommando.

Heute...

Glücklicherweise ändert der 386-Mikroprozessor dies alles. Mit seinen sechs Debugregistern bietet der 386 eingebaute Fehlersuch-Unterstützung. Sie können Breakpunkte setzen und definieren, wann sie ausgeführt werden. Die vier Debugadreßregister sind einzeln programmierbar und einzeln durch das Debugkontrollregister aktivierbar. Das Debugstatusregister führt die Debugstatusinformationen.

Zusätzlich zu den gewöhnlichen Breakpunkten unterstützt der 386-Mikroprozessor auch Datenzugriffs-Breakpunkte. Datenzugriffs-Breakpunkte sind eine große Hilfe. Diese Art von Breakpunkt tritt genau in dem Augenblick auf, in dem Daten an einer bestimmten Stelle gelesen oder beschrieben werden. Durch die Verwendung von Datenzugriffs-Breakpunkten können Sie den Befehl ausfindig machen, der eine Datenstruktur überschreibt. Der 386-Mikroprozessor gestattet Ihnen das Setzen von Breakpunkten im ROM und im RAM. Sie können eine 386-Debuggerausnahme setzen, wenn eine der folgenden Bedingungen auftritt:

- Das Ausführen der Anweisung am Breakpunkt.
- Das Ausführen jedes Befehls (Einzelschritt).
- Die Ausführung jedes Befehls an einer bestimmten Adresse.
- Das Lesen oder Schreiben eines Bytes, Worts oder Doppelworts an jeder beliebigen Adresse.
- Der Versuch, ein Debugregister zu ändern.
- Eine Task-Umschaltung zu einer bestimmten Task (nur im geschützten Modus).

Kontroll-register: LEN 3 | R/W 3 | LEN 2 | R/W 2 | LEN 1 | R/W 1 | LEN 0 | R/W 0 | * * | * | *** | GE | LE | G3 | L3 | G2 | L2 | G1 | L1 | G0 | L0 — DR7
Status-register: Reserviert von Intel | BT | BS | BD | Reserviert von Intel | B3 | B2 | B1 | B0 — DR6
Reserviert von Intel — DR5
Reserviert von Intel — DR4
Adreß-register:
Breakpunkt 3 Lineare Adresse — DR3
Breakpunkt 2 Lineare Adresse — DR2
Breakpunkt 1 Lineare Adresse — DR1
Breakpunkt 0 Lineare Adresse — DR0
* Reserviert von Intel

▶ *Bild 1: Die Debugregister des 386.*

Je komplexer die Programme und der Mikroprozessor sind, der sie ausführt, um so schwieriger gestaltet sich die Fehlersuche. In sehr komplexen Systemen können externe Debugger nicht alle Aufgaben erfüllen. Glücklicherweise ist die Fehlersuche in komplexen Programmen mit dem 386-Mikroprozessor einfacher, da dieser über eine interne Unterstützung zur Fehlersuche verfügt.

Einige Einschränkungen

Sogar mit allen oben aufgeführten Möglichkeiten können die Debugregister nicht immer einen hardwareunterstützten Debugger ersetzen. Zum Beispiel kann ein »In Circuit Emulator« (ICE) die Breakpunkte auch über einen Reset des Prozessors behalten. Der Reset löscht aber alle Debugregister des 386-Mikroprozessors, und damit auch alle vorher gesetzten Breakpunkte.

Auch können die meisten ICEs Breakpunkte im Ein-/Ausgabeadreßbereich setzen, die 386-Debugregister können aber nicht zwischen den Speicher- und Ein-/Ausgabe-Bereichen unterscheiden. Alle Datenzugriffs-Breakpunkte sind mit Speicheradressen gekoppelt. Sie brauchen einen ICE, um Ein-/Ausgabe-Bereichszugriffe zu überwachen, es sei denn, sie verwenden eine Ein-/ Ausgabe-Berechtigungs-Bitmap im virtuellen 8086-Modus.

Sie können einen hardwaregestützten Debugger wie etwa ein ICE benutzen, um einen Debugger zu debuggen. Die einzige Möglichkeit, einen 386-gestützten Debugger mit den 386-Debugregistern zu untersuchen, besteht, wenn dieser nur den Interrupt-3-Breakpunkt verwendet.

Wenn die 386-Debugregister aktiv werden, so lösen sie einen Interrupt 1 aus. Ein Debugger, der den Interrupt 1 verwendet, kann einen Debugger untersuchen, der den Interrupt 3 verwendet.

Da Sie funktionierende Hardware benötigen, um mit den 386-Debugregistern zu debuggen, ist die Fehlersuche in einem instabilen System mit einem ICE oder hardwaregestützten System erheblich einfacher. Systementwickler haben in der Regel keine funktionsfähige Tastatur, Bildschirm, Disketten und Betriebssystem, was aber Voraussetzung ist, um einen Debugger zu starten, der nur auf den 386-Debugregistern basiert. Sie benötigen ein ICE, wenn Sie Systemsoftware und Einheitentreiber untersuchen.

Sie benötigen auch mehr als nur die 386-Debugregister, um einen akzeptablen Software-Debugger zu entwickeln. Programmierer erwarten von den heutigen Debuggern, daß sie Symboltabellen auswerten können, schnell disassemblieren können und mehrere Objekt-Modulformate verstehen. Eine weitere wünschenswerte Erweiterung ist eine boolesche Vergleichsebene über dem Ausnahmebehandlungsmodul des Debuggers.

Debugger, die die Fähigkeiten des 386 ausnutzen, sind im Kommen. Die Verbindung der 386-Debugmöglichkeiten mit den Fähigkeiten, die heutige Debugger bieten, ergibt ein gutes Entwicklungswerkzeug. In der Zwischenzeit können Sie mit dem voll funktionsfähigen Programm in diesem Artikel arbeiten, das die Vorteile des 386 für einen Debugger zeigt.

Eingebaute Fähigkeiten

Der 386 verfügt über einige eingebaute Debugfähigkeiten:

• *Vier Breakpunkte.* Sie können vier Adressen angeben, die die CPU automatisch überwacht.

• *Aktivieren oder deaktivieren* der Breakpunkte. Sie können die Breakpunkte, die mit den vier Debugadresssen verbunden sind, einzeln durch verschiedene Bedingungen aktivieren oder deaktivieren.

• *Datenzugriff- und Befehls-Breakpunkte.* Sie können Breakpunkte sowohl für Datenzugriff, als auch für die Programmausführung definieren.

• *Einzelschritt.* Sie können das Programm auch Befehl für Befehl abarbeiten.

Lassen Sie uns anschauen, wie die 386-Debugfähigkeiten implementiert sind. Parallel werden wir ein Debuggerprogramm entwickeln, das diese Möglichkeiten illustriert.

Debugregister

Der 386-Mikroprozessor besitzt sechs Register, die diese Fähigkeiten kontrollieren. *Bild 1* zeigt das Format der Debugregister. Sie können auf diese durch eine Variante des MOV-Befehls zugreifen. Ein Debugregister kann entweder der Quell- oder der Zieloperand sein. Im geschützten Modus kann der MOV-Befehl, der auf die Debugregister zugreift, nur auf Privilegsebene 0 ausgeführt werden. Der Versuch, die Debugregister von einer anderen Ebene zu beschreiben, oder zu lesen, führt zu einer allgemeinen Schutzverletzung. In einem Betriebssystem, das im geschützten Modus arbeitet (wie etwa OS/2 oder XENIX) sind die Debugregister privilegierte Ressourcen, auf die nur privilegierte Tasks zugreifen können.

Debugadreßregister (DR0-DR3)

Sie können die vier Debugadreßregister mit je einem Breakpunkt laden. Jedes Register enthält eine lineare Adresse, die benutzt wird, um den Breakpunkt zu identifizieren (im nicht geschützten Modus werden die Adressen in einem Segment/Offset-Paar dargestellt).

Debugsteuerregister (DR7)

Das Debugsteuerregister gestattet das Definieren, Freigeben und Sperren der Debugbedingungen. Es gibt die Umstände an, unter denen der 386 einen Breakpunkt erkennt, den Breakpunkttyp (lokal oder global) und das Längenfeld des Breakpunkts. Die untersten 8 Bit von DR7 geben die Breakpunkte frei, oder sperren sie. Alle Breakpunkte besitzen 2 Freigabebits. Das L-Bit gibt lokale Breakpunkte frei. Das G-Bit gibt globale Breakpunkte frei. Das Debugger-Programm, das im nicht geschützten Modus arbeitet (s. unten), setzt und löscht beide Bits.

Für jede Adresse in den Registern DR0 bis DR3 geben die korrespondierenden Felder R/W0 bis RW/3 (im Register 7) die Art der Aktion an, die den Breakpunkt auslösen kann. Der 386 interpretiert diese Bits folgendermaßen:

Bits	Bedeutung
00	Breakpunkt nur bei Befehlsausführung
01	Breakpunkt nur beim schreibenden Zugriff
10	Nicht definiert
11	Breakpunkt beim Lesen und Schreiben, aber nicht bei der Befehlsausführung

Das Längenfeld des Breakpunkts wird hauptsächlich bei Datenzugriffs-Breakpunkten benötigt. Die Länge eines Datenzugriffs-Breakpunkts kann ein Byte, ein Wort oder ein Doppelwort sein. Nur die Angabe der Startadresse eines Datenelements genügt nicht, da Datenelemente drei verschiedene Längen (8, 16 oder 32 Bits) besitzen können. Das Längenfeld fügt durch die Wahl der Länge des Speicherzugriffs Flexibilität beim Auslösen des Breakpunkts hinzu. Sie können die folgenden Längen angeben:

Bits	Bedeutung
00	Länge 1 Byte
01	Länge 2 Byte (Wort)
10	Nicht definiert
11	Länge 4 Byte (Doppelwort)

Da Breakpunkte auf Befehlen nur die Adresse des ersten Bytes des Befehls angeben sollten, besitzen diese immer ein Längenfeld von 1 Byte. Wenn RWn 00 ist – ein Befehls-Breakpunkt – sollte auch LENn 00 (1 Byte) sein. Jede andere Länge ist nicht definiert.

Ist das LE- (lokal) oder GE-Bit (global) gesetzt, so verringert der 386-Mikroprozessor seine Geschwindigkeit. Dadurch können diejenigen Befehle angezeigt werden, die den Breakpunkt ausgelöst haben. Der Befehls-Breakpunkt tritt auf, bevor die entsprechende Anweisung bearbeitet wird. Ein Datenzugriffs-Breakpunkt (Lesen oder Schreiben) tritt auf, nachdem die Daten gelesen oder geschrieben wurden.

Debugstatusregister (DR6)

Der Debugger benützt die Debugstatusregister um zu erkennen, welche Debugbedingung auftrat. Der 386-Mikroprozessor setzt Statusbits, und der Debugger liest sie. Entdeckt der Mikroprozessor eine Debugausnahme, so setzt er eines oder mehrere der vier niederwertigen Bits B0 bis B3 des Statusregisters, bevor er die Debugger-Ausnahmebehandlungsroutine aufruft. Bn ist gesetzt, wenn die im Adreßregister (DRn) und im Kontrollregister (LENn und R/Wn) beschriebene Bedingung auftrat.

Das BS-Bit und das TF (Trap Flag) des 386-EFLAG-Registers ist gesetzt, wenn die Routine des Debuggers aufgerufen wird, da ein Einzelschritt-Befehl auftrat. Der Einzelschritt ist die Debugausnahme mit der höchsten Priorität. Wenn BS gesetzt ist, kann jedes der anderen Debugstatusbits auch vom 386-Mikroprozessor gesetzt worden sein. Das bedeutet, daß eine Einzelschrittausnahme zur selben Zeit auftreten kann, wie ein Befehls- oder Datenzugriffs-Breakpunkt. Die Bits BT und BS werden nur gesetzt, wenn der Mikroprozessor im geschützten Modus ist.

Das BT-Bit ist mit dem T-Bit (Debug Trap Bit) des Task-Statussegments (TSS) verknüpft. Das TSS ist eine Datenstruktur, die in der 386er Systemarchitektur definiert ist. Sie ist nur im geschützten Modus verfügbar. Jede Task besitzt ihre eigene TSS, die den Zustand des virtuellen Task-Prozessors beinhaltet. Der 386-Mikroprozessor setzt das BT-Bit bevor der Debugger-Handler aufgerufen wird, wenn eine Taskumschaltung auftrat und das T-Bit der neuen TSS-Struktur gesetzt ist. Es gibt kein zugehöriges Bit in DR7, das diesen Trap freigibt und sperrt. Das T-Bit in der TSS ist das einzige Freigabebit.

Das ICE-386 ist der In-Circuit-Emulator von Intel für den 386-Mikroprozessor. Ist das ICE angeschlossen, so hat es Vorrang über den 386-Debugger. Das BD-Bit ist gesetzt, wenn der nächste Befehl eines der Debugregister beschreibt oder liest, und das ICE-386 benutzt auch diese Register.

Der 386-Mikroprozessor löscht nur die Bits von DR6, wenn der Mikroprozessor zurückgesetzt wird. Um Verwirrung beim Identifizieren der nächsten Debugausnahme zu vermeiden, sollte der Debugger-Ausnahmebehandler sofort Nullen in DR6 schreiben.

Das Setzen von Breakpunkten

Alle vier Breakpunkte werden durch die lineare Adresse (DRn) und die Länge (LENn) definiert. Das Längenfeld gestattet Ihnen die Angabe eines 1, 2 oder 4 Byte großen Feldes. Der 386-Mikroprozessor verlangt, daß 2-Byte-Felder an einer Wortgrenze stehen (Adresse muß ein Vielfaches von 2 sein), und 4-Byte-Felder an einer Doppelwortgrenze beginnen (Adresse muß ein Vielfaches von 4 sein). Sie erhalten unerwartete Ergebnisse, wenn Befehls- oder Datenzugriffs-Breakpunkte nicht richtig ausgerichtet sind.

Sie können einen Datenzugriffs-Breakpunkt für ein nicht richtig ausgerichtetes Feld, das länger als 1 Byte ist, mit 2 oder mehr Debugadreßregistern angeben, die dann den gesamten Bereich abdecken. Jeder Eintrag muß richtig ausgerichtet sein, und die Einträge müssen die Länge des Feldes abdecken. Wenn zum Beispiel drei

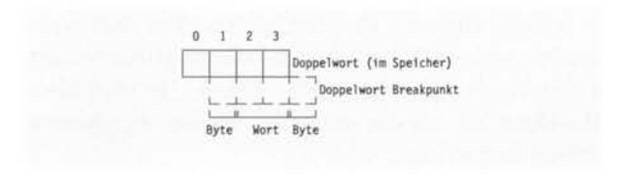


Bild 2: Ausrichten von Breakpunkt-Adressen.

Breakpunkte gesetzt sind, um ein Doppelwort, das an einer ungeraden Grenze beginnt, zu überwachen, dann müssen die Breakpunkte folgendermaßen gesetzt sein: Die erste Adresse enthält das erste Byte des Doppelworts, die zweite enthält die nächsten beiden Bytes und die dritte ist für das letzte Byte verantwortlich. *Bild 2* zeigt die richtige Ausrichtung von Breakpunkt-Adressen für die Adresse eines Doppelworts, das an einer ungeraden Grenze beginnt.

Ein Speicherzugriff löst einen Lese- oder Schreib-Datenzugriffs-Breakpunkt aus, wenn er innerhalb eines definierten Breakpunkt-Feldes liegt (das durch das Breakpunkt-Adreßregister und das zugehörige LEN-Feld angegeben ist). *Tabelle 1* zeigt Beispiele, die verdeutlichen, wann Breakpunkte auftreten und wann nicht.

Befehls-Breakpunkte haben immer die Längenangabe 1 Byte (LEN=00). Andere Werte für Befehls-Breakpunkte sind nicht definiert. Der 386 erkennt den Befehls-Breakpunkt nur, wenn die Breakpunkt-Adresse auf das erste Byte des Befehls zeigt. Besitzt ein Befehl ein Präfix, muß die Breakpunkt-Adresse auf das Präfix zeigen.

Debugausnahmen

Breakpunkte, die auf Befehle gesetzt sind, verursachen Verletzungen, alle anderen Debugbedingungen verursachen Ausnahmen. (Verletzungen werden vor der Ausführung des Befehls an dieser Adresse erkannt. Ausnahmen melden einen Datenzugriffs-Breakpunkt nachdem der Befehl ausgeführt wurde, der auf die gegebene Speicherstelle zugegriffen hat.) Die Debugausnahme kann Verletzungen und Ausnahmen zur selben Zeit melden. Im folgenden sind die vier Klassen der Debugausnahmen beschrieben.

Befehlsausführungs-Breakpunkt

Ein Befehlsausführungs-Breakpunkt ist eine Verletzung, deshalb meldet der 386 einen Befehls-Breakpunkt vor der Ausführung dieses Befehls.

Datenzugriffs-Breakpunkt

Ein Datenzugriffs-Breakpunkt ist eine Ausnahme. Der Prozessor meldet deshalb den Datenzugriffs-Breakpunkt, nachdem der Befehl ausgeführt wurde, der auf diesen Speicherbereich zugegriffen hat.

Wenn Sie Datenzugriffs-Breakpunkte verwenden, so sollten Sie LE, GE oder beide Bits von DR7 setzen. Wenn entweder das LE- oder GE-Bit gesetzt sind, so wird jede Ausnahme exakt nach der Anweisung gemeldet, die die Ausnahme ausgelöst hat. Diese genaue Angabe wird durch das

	DR0 Adresse: 0A0001 (Länge 1)	**DR1 Adresse:** 0A0002 (Länge 1)	**DR2 Adresse:** 0B0002 (Länge 2)	**DR3 Adresse:** 0C0000 (Länge 4)
Ausnahme-adressen	0A0001 (Länge 1)	0A0002 (Länge 1)	0B0002 (Länge 2)	0C0000 (Länge 4)
	0A0001 (Länge 2)	0A0002 (Länge 2)	0B0001 (Länge 4)	0C0003 (Länge 1)
Keine Ausnahme	0A0000 (Länge 1)	0A0003 (Länge 4)	0B0000 (Länge 2)	0C0004 (Länge 4)

Tabelle 1: Breakpunkt-Adreßbeispiele.

Verlangsamen erreicht. Die 386er Ausführungseinheit wartet auf die vollständige Übertragung des Datenoperanden, bevor die neue Anweisung bearbeitet wird. Ist weder GE noch LE gesetzt, so können Datenzugriffs-Breakpunkte erste eine Anweisung nach der auslösenden Anweisung oder gar nicht gemeldet werden. Der 386 überlappt normalerweise die Ausführung der Befehle mit Speicherzugriffen in einem solchen Maße, daß die Ausführung des nächsten Befehls beginnen kann, bevor die Speicherzugriffe des vorhergehenden Befehls beendet sind.

Erzeugt der Debugger einen Datenschreib-Breakpunkt, so sollte er die Originaldaten vor dem Setzen des Breakpunkts retten. Da die Datenzugriffs-Breakpunkte Ausnahmen sind, wird das Verändern der Daten durchgeführt, bevor die Änderung gemeldet worden ist. Die Behandlungsroutine kann die Originaldaten (gerettet) anzeigen, nachdem der Breakpunkt ausgelöst wurde. Die Daten im Debugregister können für die Adressierung des neuen Werts, der durch die Anweisung in den Speicher geschrieben wurde, die den Breakpunkt auslöste, hergenommen werden.

Einzelschritt-Ausnahme

Diese Debugbedingung tritt am Ende einer Anweisung auf, wenn am Beginn dieser Anweisung das Trap Flag (TF) im Flagregister den Wert 1 enthielt. Diese Ausnahme tritt nicht bei der Anweisung auf, die das TF-Flag setzt. Wenn z.B. die Anweisung POPF das TF-Flag setzt, tritt der Trap erst am Ende des nächsten Befehls auf.

Die Interrupt-Prioritäten gewährleisten, daß bei einem externen Interrupt der Einzelschrittmodus abgeschaltet ist. Treten sowohl ein externer, als auch ein Einzelschritt-Interrupt zusammen auf, wird zuerst der Einzelschritt-Interrupt bearbeitet. Dieser löscht das TF-Flag. Nach dem Retten der Rücksprungadresse oder der Taskumschaltung wird die Eingabe des externen Interrupts vor der ersten Anweisung der Einzelschritt-Behandlungsroutine behandelt. Liegt der externe Interrupt immer noch an, so wird er zuerst bedient. Der externe Interrupt-Handler wird nicht im Einzelschrittmodus ausgeführt. Wenn Sie einen externen Interrupt im Einzelschritt bearbeiten wollen, müssen Sie eine Anweisung INT n, die auf einen Interrupt-Handler zeigt, im Einzelschrittmodus bearbeiten.

▸ *Tabelle 2: Die sieben Breakpunkt-Bedingungen des Interrupt 1.*

Statusregister-Flags	Bedingung
BS=1	Einzelschritt Ausnahme
B0=1 AND (GE0=1 OR LE0 =1)	Breakpunkt DR0, LEN0, R/W0
B1=1 AND (GE1=1 OR LE1=1)	Breakpunkt DR1, LEN1, R/W1
B2=1 AND (GE2=1 OR LE2=1)	Breakpunkt DR1, LEN1, R/W1
B3=1 AND (GE3=1 OR LE3=1)	Breakpunkt DR3, LEN3, R/W3
BD=1	Debugregister sind nicht verfügbar. ICE-386 benötigt die Debugregister (nur im geschützten Modus)
BT=1	Taskumschaltung (nur im geschützten Modus)

Taskumschaltungs-Breakpunkt
Im geschützten Modus tritt nach der Umschaltung zu einer neuen Task eine Unterbrechung ein, wenn das TSS-Bit gesetzt ist. Der Breakpunkt ist nach der Taskumschaltung aktiv, aber bevor die erste Anweisung ausgeführt wird. Die Ausnahme-Behandlungsroutine kann die Taskumschaltung am BT-Flag des Debugstatusregisters (DR6) erkennen.

Interrupts

Sowohl der 386, als auch frühere Mikroprozessoren verwenden zwei Interruptvektoren für das Debuggen. Interrupt 1 ist für die Einzelschrittausnahme reserviert, Interrupt 3 für Befehls-Breakpunkte. Zusätzlich erzeugt der 386-Mikroprozessor einen Interrupt, wenn auf ein Debugregister zugegriffen wird.

Interrupt 1
Die Behandlungsroutine für den Interrupt 1 ist normalerweise ein Debugger, oder Teil eines Debugsystems. *Tabelle 2* zeigt die sieben Breakpunkt-Bedingungen, die einen Interrupt 1 auslösen können. Der Debugger kann die Flags in DR6 und DR7 bewerten, um zu erkennen, welche Bedingung zum Auslösen der Ausnahme geführt hat.

Interrupt 3
Diese Ausnahme wird durch die Ausführung einer INT-3-Anweisung ausgelöst. Ein typischer Debugger, der nicht von den erweiterten 386-Mikroprozessor-Fähigkeiten Gebrauch machte, setzte einen Breakpunkt durch den Ersatz des ersten Bytes des Befehls mit einer INT-3-Anweisung.

Früher verwendeten Mikroprozessoren die Breakpunkt-Ausnahme für das Abfangen von Anweisungen. Der 386-Mikroprozessor löst dieses Problem eleganter durch die Debugregister und den Interrupt 1. Die Breakpunkt-Ausnahme ist aber noch immer nützlich, um Debugger zu debuggen, da die Ausnahme-Behandlungsroutine in einem anderen Programm, als dem Debugger enthalten sein kann. Die Breakpunkt-Ausnahme kann auch nützlich sein, wenn Sie mehrere Breakpunkte als die vier mit den Debugregistern möglichen setzen.

Ein Debugger-Beispielprogramm

Sie brauchen Ihren alten Debugger nicht aufzugeben, um die Vorteile der 386-Debugmöglichkeiten zu nutzen. Mit Hilfe eines netten Debugger-Hilfsprogramms (wie das funktionsfähige Programmfragment in diesem Artikel), können Sie Ihren Debugger weiter verwenden. Durch Setzen und Löschen von Datenzugriffs-Breakpunkten mit einer Popup-Routine, ergänzt das gezeigte Debugger-Hilfsprogramm den Debugger. Sie können die Vorteile Ihres Debuggers nutzen (zum Beispiel Symbole und Disassemblierung) und verfügen zusätzlich über die im 386 eingebauten Fehlersuch-Unterstützungen.

Das 386-gestützte Debugger-Hilfsprogramm läuft im Hintergrund als Unterstützung des anderen Debuggers. Dieses Hilfsprogramm ist ein Hintergrundprogramm (TSR), das durch Drücken der Taste [SysReq] oder durch eine Debugausnahme aktiviert wird. Der Popup-Bildschirm beschreibt die 386-Register und zeigt an, welcher der vier individuell programmierbaren Breakpunkte auftrat.

Unsere Debugger-Hilfe bietet einige Besonderheiten. Eine Bildschirmmaske zeigt alle Register, die Sie benötigen, und gestattet Ihnen das einfache Eingeben von Breakpunkt-Adressen, das Aktivieren und Deaktivieren der Breakpunkte sowie das Definieren der Breakpunkt-Bedingungen. *Bild 3* zeigt ein Beispiel dieser Anzeige, die in die folgenden fünf Bereiche aufgeteilt ist:

- BREAKPOINT enthält die vier 386-Debugregister und ihre Optionen.
- COMPARE zeigt die verfügbaren booleschen Vergleichsoperationen.
- SPECIAL REGS zeigt das 386-EFLAGS-Register in hexadezimaler Darstellung, das Debugsteuerregister (DR7) und das Debugstatusregister (DR6).
- REGISTER SET zeigt die Werte der am meisten benutzten 386-Register.
- EFLAGS zeigt die EFLAGS-Register dekodiert.

Sie können die meisten der angezeigten Werte ändern, indem Sie entweder editieren oder einfach im entsprechenden Feld umschalten.

Eine boolesche Vergleichsebene entscheidet, ob die Ausnahme den spezifizierten Bedingungen entspricht. Das Debugger-Hilfsprogramm verfügt zusätzlich über eine Vergleichslogik, die an- und abgeschaltet werden kann, indem das Umschaltfeld (sw) in COMPARE geändert wird. Sie können auch die booleschen Felder (bool) in COMPARE in folgende Zustände umschalten:

<	kleiner
<=	kleiner oder gleich
=	gleich
<>	ungleich
>	größer
>=	größer oder gleich

Der verwendete Wert in der Vergleichslogik wird durch das Ändern des Value-Feldes in COMPARE gewählt.

Eine Interrupt-Serviceroutine behandelt die Ausnahmen. Das Debugger-Hilfsprogramm verfügt über zwei Interrupt-Serviceroutinen (ISRs). Eine ISR behandelt den SysReq-Tastendruck, die andere behandelt die Debugger-Ausnahme-Interrupts. Ein Teil des Programms in *Listing 1* beinhaltet eine Interrupt-Serviceroutine.

Von den Interrupt-Serviceroutinen wird ein gemeinsamer Programmteil aufgerufen. Im Debugger-Hilfsprogramm verwenden beide ISRs ein gemeinsames Programmteil, das die 386-Register in eine große Datenstruktur kopiert. Dieser Programmteil übergibt dann die Adresse der Datenstruktur an das Hochsprachenprogramm, das die Aufgaben der Benutzerschnittstellen erledigt.

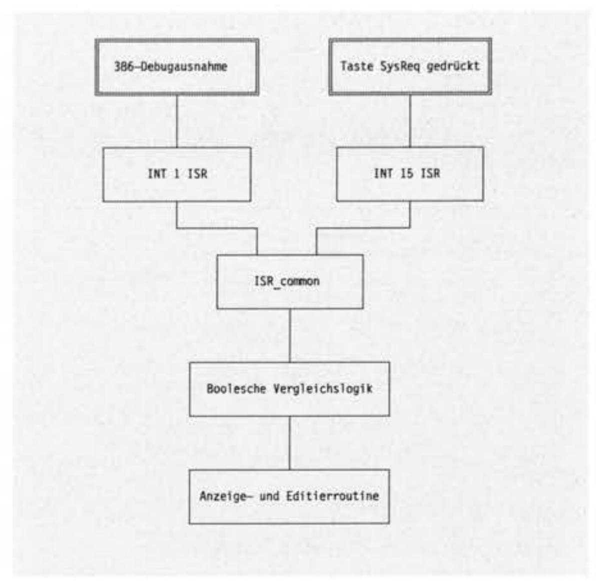
Nachdem die Register (vom Anwender) geändert wurden, übergibt das Hochsprachenprogramm die Kontrolle wieder an die gemeinsame Routine. Diese kopiert ihrerseits das Registerabbild wieder in die 386-Register und »legt sich schlafen« bis die nächste Ausnahme auftritt, oder die SysReq-Taste betätigt wird. *Listing 1* beinhaltet den gemeinsamen Programmteil in Assembler.

Einige der Assemblerbefehle werden für Sie ungewohnt sein. Da es noch wenige 386-Assembler gibt, habe ich die 386-spezifischen Befehle mit der Hand assembliert. Viele sind sehr einfach zu kodieren, da sie nur ein Präfix-Byte benötigen. Dieses Präfix-Byte gibt an, daß der folgende Befehl mit 32-Bit-Operanden, statt mit 16-Bit-Operanden arbeitet, wie die Befehle, die es vor dem 386 gab. Wenn Sie zum Beispiel das Präfix-Byte 066H vor die Anweisung

```
MOV     AX,BX
```

setzen, ergibt dies eine 386-Anweisung, die der Mikroprozessor als folgenden Befehl behandelt:

```
MOV     EAX,EBX
```



◄ *Bild 4: Programmfluß des Debugger-Hilfsprogramms.*

Bild 4 zeigt den Programmfluß des Debugger-Hilfsprogramms.

Das Verständnis der Debugunterstützung des 386-Mikroprozessors, wie ich sie in diesem Artikel dargestellt habe, ergibt einen guten Ausgangspunkt zum Finden von schwierigen Fehlern. Es setzt die Basis für das Programmieren spezieller Debughilfsmittel, die zu dem gegebenen Problem die Suche der Lösung erleichtern. Ich will hiermit aber keinen Leser verleiten, einen Debugger zu programmieren. Aber unter gewissen Umständen verkürzt ein Debugger-Hilfsprogramm wie das hier vorgestellte die Fehlersuche. Außerdem gibt es Ihnen die Befriedigung, die Debugfähigkeiten besser auszunutzen und das interne Arbeiten des modernsten Mikroprozessors besser zu verstehen.

Marion Hansen, Nick Stuecklen

```
             B R E A K P O I N T                C O M P A R E
  sw segm  offset      type    lenght          sw bool  value

  ■  0000  00000000    code    byte                 =   00000000       SPECIAL REGS
     0000  00000000    code    byte                 =   00000000    DR7
     0000  00000000    code    byte                 =   00000000    DR6
     0000  00000000    code    byte                 =   00000000    EFLAGS

                  R E G I S T E R   S E T                      E F L A G S
CS:EIP 000B 00001372  EAX 00001372  EBP 00000000     overflow   no aux carry
CS:ESP 185A 000022ED  EBX 000022ED   FS 0000         forward    odd parity
DS:ESI 185A 00000000  ECX 00000000   GS 0000         positive   carry
ES:EDI 185A 00000000  EDX 00000000  CR0 FFFFFFF1     zero       disable ints

  ←→↑↓  F1=help   F3=INT3   F10=exit   F9=app screen  F5=toggle  ←(rubout)=edit
```

Bild 3: Das Debugger-Hilfsprogramm zeigt die 386-Mikroprozessor-Register mit den vier Debugregistern in der linken oberen Ecke. Es erlaubt Ihnen die Breakpunkte zu setzen und die Bedingungen hierfür zu definieren.

Listing 1: Das Debugger-Hilfsprogramm.

```
NAME    ISR

EXTRN display_and_edit_regs:FAR

CODE SEGMENT PUBLIC 'CODE'
ASSUME CS:CODE
PUBLIC INT1_ISR, INT9_ISR, SYSREQ_ISR, ISR_common
PUBLIC orig_INT1_ISR_ptr, orig_INT3_ISR_ptr, orig_SYSREQ_ISR_ptr
PUBLIC orig_INT9_ISR_ptr

$EJECT
;--------------------------------------------------------------------------
;
;  This Assembly language module is composed of two interrupt service routines
;  (ISRs) and another block of code shared between the two ISRs.
;  The module is used to either awaken a register display/edit routine as a
;  result of an 80386 debug exception, or to awaken the display/edit routine
;  as a result of a user request to "pop-it-up" (via the SYSREQ key).
;
;  The first ISR, known as INT1_ISR, services the 80386 debug exception interrupt,
;  and then CALLs the common block of code, ISR_common.
;
;  ISR_common copies the current 80386 register image into a large data
;  structure and then CALLs the register display/edit routine (written in a
;  higher-level language).
;
;  The second ISR, known as SYSREQ_ISR, is chained into the BIOS INT 15h
;  interrupt, and it merely awaits a SYSREQ key press (passing any other
;  INT 15h requests to the original INT 15h handler) before calling
;  ISR_common.
;
;  The display/edit routine (not shown in this article, but written in
;  PL/M-86) allows the user to examine and/or modify, by reading/altering
;  the aforementioned register image data structure, the normal 80386
;  registers (in full 32 bit form), as well as the debug registers.
;
;--------------------------------------------------------------------------

$EJECT
;--------------------------------------------------------------------------
;
;  Define structure/data area used to hold copies of the register images
;  as they exist when the INT 1 ISR is entered.
;
;  W A R N I N G :
;       The register ordering is  F I X E D ! ! !  The display_and_edit_regs
;       subroutine assumes a predefined ordering.
;
;  A L S O :
;       You probably noticed that each table entry is 32 bits long,
;       even though some registers are actually only 16 bits long (like CS).
;       The uniform entry size makes indexing into the table much easier
;       on the display routines.  If a register is only 16 bits wide, then
;       the high order 16 bits of its register image from the following structure
;       are wasted.  This is not a problem, since the program is not so large
;       that such wastage is critical.
;
;--------------------------------------------------------------------------
ISR_register_image      LABEL   DWORD
ISR_DR0         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_DR1         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_DR2         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_DR3         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_DR6         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_DR7         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_CS          DW      0                       ;This register is 16 bits wide
                DW      ?                       ;  (these 16 bits unused)
ISR_EIP         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_SS          DW      0                       ;This register is 16 bits wide
                DW      ?                       ;  (these 16 bits unused)
ISR_ESP         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_DS          DW      0                       ;This register is 16 bits wide
                DW      ?                       ;  (these 16 bits unused)
ISR_ESI         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_ES          DW      0                       ;This register is 16 bits wide
                DW      ?                       ;  (these 16 bits unused)
ISR_EDI         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_EAX         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_EBX         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_ECX         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_EDX         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_EBP         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_FS          DW      0                       ;This register is 16 bits wide
                DW      ?                       ;  (these 16 bits unused)
ISR_GS          DW      0                       ;This register is 16 bits wide
                DW      ?                       ;  (these 16 bits unused)
ISR_CR0         DW      0                       ;This register is 32 bits wide
                DW      0
ISR_EFLAGS      DW      0                       ;This register is 32 bits wide
                DW      0
;--------------------------------------------------------------------------
;
;  The following space is allocated for the display_and_edit_regs routine,
;  but is not used by the ISRs.
;
;--------------------------------------------------------------------------
dr0_segment             DD      0
dr0_offset              DD      0
dr0_compare_value       DD      0
dr0_compare_enable      DD      0
dr1_segment             DD      0
dr1_offset              DD      0
dr1_compare_value       DD      0
dr1_compare_enable      DD      0
dr2_segment             DD      0
dr2_offset              DD      0
dr2_compare_value       DD      0
dr2_compare_enable      DD      0
dr3_segment             DD      0
dr3_offset              DD      0
dr3_compare_value       DD      0
dr3_compare_enable      DD      0
dr0_boolean             DD      0
dr1_boolean             DD      0
dr2_boolean             DD      0
dr3_boolean             DD      0
```

```
$EJECT

;--------------------------------------------------------------------------
;
;  Allocate storage for INT 3 flag — this flag is set/reset by the
;  register display routine, and indicates whether the ISR_common code
;  should trigger an INT3 shortly before RETing.
;
;--------------------------------------------------------------------------
INT3_flag               DB      ?


;--------------------------------------------------------------------------
;
;  Allocat storage for the request flag — we set this flag to indicate
;  to the register display routine whether we're calling it from
;  SYSREQ ("pop-up" request) or INT1 (debug exception has occurred).
;
;--------------------------------------------------------------------------
request_flag            DB      ?
INT1_request            EQU     0
SYSREQ_request          EQU     1


;--------------------------------------------------------------------------
;
;  Define the values necessary for processing SYSREQ key presses.
;
;--------------------------------------------------------------------------
SYSREQ_key_pressed              EQU     08500h
keyboard_status_port            EQU     064h
input_buffer_full               EQU     002h
enable_keyboard                 EQU     0AEh
PIC                             EQU     020h
EOI                             EQU     020h


;--------------------------------------------------------------------------
;
;  ISR local stack definition...
;
;--------------------------------------------------------------------------
                DW      512 DUP (0)
ISR_stack       LABEL   WORD

;--------------------------------------------------------------------------
;
;  Where we store the original SS:SP before we install the local stack.
;
;--------------------------------------------------------------------------
orig_ISR_stack_ptr      DW      ?
                        DW      ?

;--------------------------------------------------------------------------
;
;  Storage for copy of local code segment value.
;
;--------------------------------------------------------------------------
local_data_seg          DW      CODE

;--------------------------------------------------------------------------
;
;  Define storage for the original interrupt service routine addresses
;  for the interrupts onto which we'll be chaining or installing ourselves.
;
;--------------------------------------------------------------------------
orig_INT1_ISR_ptr       DW      ?
                        DW      ?
orig_INT3_ISR_ptr       DW      ?
                        DW      ?
orig_INT9_ISR_ptr       DW      ?
                        DW      ?
orig_SYSREQ_ISR_ptr     DW      ?
                        DW      ?

;--------------------------------------------------------------------------
;
;  The following instruction prefix is 80386-specific.  It identifies the
;  subsequent instruction as one which uses a 32 bit operand size.  This
;  prefix allows us to create 80386 instructions using an 80286 assembler.
;
;--------------------------------------------------------------------------
operand_size_32_prefix  EQU     066H

;--------------------------------------------------------------------------
;
;  Define stack frame which exists at the time the ISR_common is CALLed.
;
;--------------------------------------------------------------------------
ISR_stack_parm          STRUC
        ISR_RTNA        DW      ?       ;Return address (to INT1_ISR or SYSREQ_ISR)
        old_IP          DW      ?       ;CS:IP of code which was in progress at time
        old_CS          DW      ?       ;  the debug exception occurred
        old_FLAGS       DW      ?       ;State of the FLAGS at time of exception
ISR_stack_parm          ENDS

;--------------------------------------------------------------------------
;
;  Define flag which can be used to determine whether or not we're already
;  servicing an interrupt request (in other words, are we being re-entered ? )
;
;--------------------------------------------------------------------------
ISR_in_progress         DB      FALSE
FALSE                   EQU     0
TRUE                    EQU     0FFh

$EJECT

;--------------------------------------------------------------------------
;
;               INT1 (80386 Debug Exception) Handler
;
;  This interrupt service routine can be entered as a result of
;  one of the following conditions:
;
;               1 - instruction execution breakpoint
;               2 - data access breakpoint
;               3 - general detect fault
;               4 - single step trap
;               5 - task switch breakpoint
;
;  The ISR first ensures that it is not being re-entered, and then
;  CALLs ISR_common.
;
;--------------------------------------------------------------------------

INT1_ISR        PROC    FAR

        JMP     INT1_start                      ;Jump around header field
        DB      'DAGGER'                        ;This header is a "marker" used to
                                                ;  determine if the ISR is already
                                                ;  installed
```

Listing 1: (Fortsetzung)

```
INT1_start:
        PUSHF                                   ;Save FLAGs
        CMP     CS:ISR_in_progress,TRUE         ;Are we being re-entered?
        JNE     not_being_reentered             ;NO
        POPF                                    ;YES, recover FLAGs,
        STI                                     ;  put interrupts back on,
        IRET                                    ;  and leave


;-----------------------------------------------------------------------
;
;  Mark "in progress" flag so that we can't be re-entered
;
;-----------------------------------------------------------------------
not_being_reentered:
        MOV     CS:ISR_in_progress,TRUE         ;Show "in progress"
        MOV     CS:request_flag,INT1_request    ;Set flag indicating that this
                                                ;  INT occurred as a result of
                                                ;  an 80386 debug exception
        POPF                                    ;Recover FLAGS

        CALL    ISR_common                      ;CALL common ISR code
        IRET                                    ;  and leave

INT1_ISR        ENDP

$EJECT

;-----------------------------------------------------------------------
;
;               SYSREQ (System Request Key) Handler
;
;  This interrupt service routine can be entered as a result of
;  one of the following conditions:
;
;               SYSREQ key pressed
;                       OR
;               some other BIOS INT 15 request occurred
;
;  We will only CALL ISR_common if the ISR was entered as a result of a user
;  request to "pop-up" the register display/edit screen.
;  Otherwise, we simply chain onto the old BIOS INT 15 ISR.
;
;-----------------------------------------------------------------------
SYSREQ_ISR      PROC    FAR

        PUSHF                                   ;Save FLAGS
        CMP     AX,SYSREQ_key_pressed           ;Was the INT for SYSREQ key ?
        JE      process_SYSREQ                  ;YES — do local processing

chain_to_original_ISR:
        POPF                                    ;NO, recover FLAGS
        STI                                     ;Interrupts back on
        JMP     DWORD PTR CS:orig_SYSREQ_ISR_ptr ;Chain on to original SYSREQ ISR

process_SYSREQ:
        PUSH    AX                              ;Save AX
        MOV     AL,EOI
        OUT     PIC,AL                          ;Issue End-of-Interrupt to PIC
await_keybd_controller:
        IN      AL,keyboard_status_port         ;Get keyboard controller status
        TEST    AL,input_buffer_full  ;Keyboard controller ready to accept command ?
        JNZ     await_keybd_controller          ;NO
        MOV     AL,enable_keyboard              ;YES
        OUT     keyboard_status_port,AL         ;Re-enable keyboard
        POP     AX                              ;  recover AX
        POPF                                    ;  recover FLAGS,

;-----------------------------------------------------------------------
;
;  IF we're being re-entered, then simply chain to original ISR.
;
;-----------------------------------------------------------------------
        PUSHF                                   ;Save flags
        CMP     CS:ISR_in_progress,TRUE         ;Are we being re-entered?
        JNE     SYSREQ_not_being_reentered      ;NO
        JMP     chain_to_original_ISR           ;Chain to original ISR

;-----------------------------------------------------------------------
;
;  ELSE:  mark "in progress" flag so that we can't be re-entered
;
;-----------------------------------------------------------------------
SYSREQ_not_being_reentered:
        MOV     CS:ISR_in_progress,TRUE         ;Show "in progress"
        MOV     CS:request_flag,SYSREQ_request  ;Set flag indicating that this
                                                ;  INT occurred as a result of
                                                ;  a user request to "pop-up"
                                                ;  the display/edit routine
        POPF                                    ;Recover FLAGS

        CALL    ISR_common                      ;  CALL common ISR code,
        JMP     DWORD PTR CS:orig_SYSREQ_ISR_ptr; and then JMP to original INT 15h

SYSREQ_ISR      ENDP

$EJECT

INT9_ISR        PROC    FAR

        JMP     DWORD PTR CS:orig_INT9_ISR_ptr

INT9_ISR        ENDP

$EJECT

ISR_common      PROC    NEAR

;-----------------------------------------------------------------------
;
;  Common interrupt service routine code (shared by INT1_ISR and SYSREQ_ISR)
;
;  This common block of code simply copies the current register state into
;  the large data structure described earlier.  The address of the data
;  structure is passed to the display/edit registers routine, which is a
;  higher-level language subroutine that allows the user to edit the
;  normal 80386 registers, as well as edit the debug registers.
;
;-----------------------------------------------------------------------
        STI                                     ;Interrupts back on

;-----------------------------------------------------------------------
;
;  Put EBP into data structure.
;  Put EAX into data structure.
;  Then get EFLAGS into data structure via EAX.  FLAGS (low word of EFLAGS)
;  were pushed onto the stack by the CPU when the INT occurred.  We'll
;  just OR the FLAGS (low 16 bits of EFLAGS) from the stack with the high word
;  of current EFLAGS.
;
;-----------------------------------------------------------------------
        DB      operand_size_32_prefix
        MOV     CS:ISR_EBP,BP                   ;EBP into global structure

        DB      operand_size_32_prefix
        MOV     BP,SP                           ;EBP = current ESP

        DB      operand_size_32_prefix
        MOV     CS:ISR_EAX,AX                   ;EAX into global structure

        DB      operand_size_32_prefix
        PUSHF                                   ;PUSH EFLAGS
        DB      operand_size_32_prefix
        POP     AX              ;POP EAX (high word of EAX has high word of EFLAGS)
        MOV     AX,[BP].old_FLAGS       ;Get FLAGS from stack into low word of EAX
        DB      operand_size_32_prefix
        MOV     CS:ISR_EFLAGS,AX                ;EFLAGS into global structure

$EJECT

;-----------------------------------------------------------------------
;
;  Get CS:IP from stack (placed there by 80386 when INT 1 occurred).
;
;-----------------------------------------------------------------------
        MOV     AX,[BP].old_CS                  ;Get CS from stack
        MOV     CS:ISR_CS,AX                    ;CS into global structure
        DB      operand_size_32_prefix
        SUB     AX,AX                           ;Clear high word of EAX
        MOV     AX,[BP].old_IP          ;Get IP from stack into low word of EAX
        DB      operand_size_32_prefix
        MOV     CS:ISR_EIP,AX                   ;EIP into global structure

;-----------------------------------------------------------------------
;
;  Copy the rest of the registers into the data structure.
;
;-----------------------------------------------------------------------
        MOV     CS:ISR_DS,DS                    ;DS

        DB      operand_size_32_prefix
        MOV     CS:ISR_ESI,SI                   ;ESI

        MOV     CS:ISR_ES,ES                    ;ES

        DB      operand_size_32_prefix
        MOV     CS:ISR_EDI,DI                   ;EDI

        DB      operand_size_32_prefix
        MOV     CS:ISR_EBX,BX                   ;EBX

        DB      operand_size_32_prefix
        MOV     CS:ISR_ECX,CX                   ;ECX

        DB      operand_size_32_prefix
        MOV     CS:ISR_EDX,DX                   ;EDX

        DB      08Ch,0E0h                       ;FS
        MOV     CS:ISR_FS,AX

        DB      08Ch,0E8h                       ;GS
        MOV     CS:ISR_GS,AX

        DB      00Fh,020h,0C0h                  ;CR0
        DB      operand_size_32_prefix
        MOV     CS:ISR_CR0,AX

        DB      0Fh,021h,0C0h                   ;DR0
        DB      operand_size_32_prefix
        MOV     CS:ISR_DR0,AX

        DB      0Fh,021h,0C8h                   ;DR1
        DB      operand_size_32_prefix
        MOV     CS:ISR_DR1,AX

        DB      0Fh,021h,0D0h                   ;DR2
        DB      operand_size_32_prefix
        MOV     CS:ISR_DR2,AX

        DB      0Fh,021h,0D8h                   ;DR3
        DB      operand_size_32_prefix
        MOV     CS:ISR_DR3,AX

        DB      0Fh,021h,0F0h                   ;DR6
        DB      operand_size_32_prefix
        MOV     CS:ISR_DR6,AX

        DB      0Fh,021h,0F8h                   ;DR7
        DB      operand_size_32_prefix
        MOV     CS:ISR_DR7,AX

$EJECT

;-----------------------------------------------------------------------
;
;  Save original SS:SP into save area and also into the global structure,
;  and then create a new SS:SP so that we can CALL a stack-intensive
;  P/LM-86 routine.
;
;-----------------------------------------------------------------------
        DB      operand_size_32_prefix
        ADD     BP,SIZE ISR_stack_parm          ;Adjust EBP (which is a copy
                                                ;  of original ESP) so that it
                                                ;  reflects original state of
                                                ;  ESP at the time INT occurred
        DB      operand_size_32_prefix
        MOV     CS:ISR_ESP,BP                   ;ESP into global structure
        MOV     CS:ISR_SS,SS                    ;SS into global structure

        MOV     CS:orig_ISR_stack_ptr,SP        ;Save SP into local storage
        MOV     CS:orig_ISR_stack_ptr + 2,SS    ;Save SS into local storage

        CLI                             ;Clear INTs while working on stack regs
        MOV     SS,CS:local_data_seg            ;New SS
        LEA     SP,ISR_stack                    ;New SP
        STI                                     ;Restore INTs

;-----------------------------------------------------------------------
;
;  CALL the PL/M-86 procedure responsible for displaying and processing
;  the information we've just put into the data structure.
;  The PL/M routine will read and display the register state (as shown
;  in the data structure), and will allow the user to indirectly modify the
;  registers (including the debug registers) via edits to that same
;  structure.
;  When the display/edit routine RETurns, we'll copy the register image
;  back into the 80386 registers.
;
;-----------------------------------------------------------------------
        LEA     AX,ISR_register_image           ;Pass the address of the
        PUSH    CS                              ;  register image
        PUSH    AX                              ;  as a pointer on the stack
```

```
        LEA     AX,INT3_flag                    ;Pass the addrs of INT3 flag
        PUSH    CS                              ;  so that the display/edit routine
        PUSH    AX                        ; can set/reset it as the user so desires
        MOV     AL,CS:request_flag              ;Pass the request type flag so
        PUSH    AX                              ;  that the display/edit routine
                                          ; can determine whether it was called
                                          ; as the result of an INT 1 or SYSREQ
        CALL    display_and_edit_regs           ;CALL the P/LM-86 PROCedure
$EJECT
;------------------------------------------------------------------------
;
;  Transfer the edited register images from the global structure
;  back into the 80386 registers.
;
;------------------------------------------------------------------------
        DB      operand_size_32_prefix
        MOV     BX,CS:ISR_EBX                   ;EBX

        DB      operand_size_32_prefix
        MOV     CX,CS:ISR_ECX                   ;ECX

        DB      operand_size_32_prefix
        MOV     DX,CS:ISR_EDX                   ;EDX

        DB      operand_size_32_prefix
        MOV     SI,CS:ISR_ESI                   ;ESI

        DB      operand_size_32_prefix
        MOV     DI,CS:ISR_EDI                   ;EDI

        DB      operand_size_32_prefix
        MOV     BP,CS:ISR_EBP                   ;EBP

        MOV     ES,CS:ISR_ES                    ;ES

        MOV     DS,CS:ISR_DS                    ;DS

        MOV     AX,CS:ISR_FS
        DB      08Eh,0E0h                       ;MOV FS,AX

        MOV     AX,CS:ISR_GS
        DB      08Eh,0E8h                       ;MOV GS,AX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_CR0
        DB      00Fh,022h,0C0h                  ;MOV CR0,EAX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_DR0
        DB      00Fh,023h,0C0h                  ;MOV DR0,EAX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_DR1
        DB      00Fh,023h,0C8h                  ;MOV DR1,EAX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_DR2
        DB      00Fh,023h,0D0h                  ;MOV DR2,EAX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_DR3
        DB      00Fh,023h,0D8h                  ;MOV DR3,EAX

;------------------------------------------------------------------------
;
;  Clear out DR6 — all bits in that reg must be reset after each
;  INT 1  (ignore whatever is currently sitting in the DR6 register image).
;
;------------------------------------------------------------------------
        DB      operand_size_32_prefix
        SUB     AX,AX                           ;SUB EAX,EAX
        DB      00Fh,023h,0F0h                  ;MOV DR6,EAX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_DR7                   ;MOV EAX,CS:ISR_DR7
        DB      00Fh,023h,0F8h                  ;MOV DR7,EAX

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_EFLAGS                ;MOV EAX,CS:ISR_EFLAGS
        DB      operand_size_32_prefix
        PUSH    AX                              ;PUSH EAX
        DB      operand_size_32_prefix
        POPF                                    ;POP EFLAGS

        DB      operand_size_32_prefix
        MOV     AX,CS:ISR_EAX                   ;MOV EAX,CS:ISR_EAX

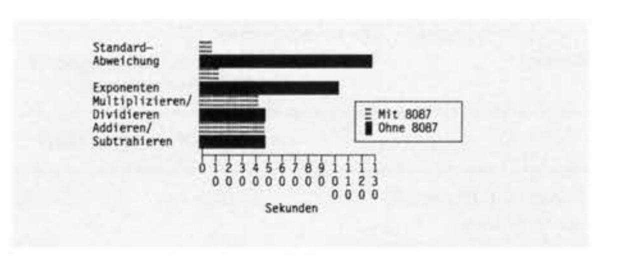
$EJECT

;------------------------------------------------------------------------
;
;  Clean up stack.
;
;------------------------------------------------------------------------
        CLI                                     ;Clear INTs while working on stack
        MOV     SS,CS:orig_ISR_stack_ptr + 2    ;Get original SS
        MOV     SP,CS:orig_ISR_stack_ptr        ;Get original SP
        MOV     CS:ISR_in_progress,FALSE        ;Show "no longer in progress"
;------------------------------------------------------------------------
;
;  Issue an INT3 (old-style debugger interrupt) if user so directed.
;  In that fashion, we can trigger a "real" debugger (presumably one which
;  will allow us to examine/modify memory and display symbol information).
;  We can merely get rid of the local caller's return address,
;  and then "JMP" directly to the original INT3 ISR.  Since our code
;  was entered as a result of an INT1, then the stack will already be
;  set up such that the INT3 routine has only to execute an IRET.
;
;------------------------------------------------------------------------
        PUSHF
        CMP     CS:INT3_flag,0                  ;Did the user want an INT3 ?
        JZ      ISR_common_RET                  ;NO
        POPF                                    ;YES, recover flags,
        ADD     SP,2                            ;  pop-off the local
                                                ;  caller's return address
        PUSH    BP                              ;Save reg
        MOV     BP,SP                           ;Get stack ptr
        INC     WORD PTR [BP + 2]               ;Adjust IP on stack such that
                                                ;  the INT3 handler can DEC to
                                                ;  adjust for the "INT3"
        POP     BP                              ;Recover reg
        STI                                     ;Put interrupts back on,
        JMP     DWORD PTR CS:orig_INT3_ISR_ptr  ;  and then "JMP" to the
                                                ;  original INT3 ISR
ISR_common_RET:
        POPF                                    ;Recover flags
        STI                                     ;Interrupts back on
        RET
ISR_common      ENDP
CODE    ENDS
END
```

Programmgeschwindigkeiten erhöhen mit Mathematik-Coprozessoren:

Auf, auf und davon

Der Mikroprozessor in Ihrem Computer ist leistungsfähig, aber nicht für komplexe arithmetische Operationen ausgelegt. Wenn es sich um einen 8086, 8088, 80286 oder 80386 handelt, so kann er Fließkommaberechnungen erheblich schneller und präziser ausführen, wenn er zusätzlich über einen mathematischen Coprozessor verfügt. Die Coprozessoren sind auch für kaufmännische Applikationen nützlich. Der Coprozessor kann binär kodierte Dezimalzahlen bis zu einer Länge von 18 Stellen im Zahlenbereich 2×10^{-9} bis 2×10^{9} ohne Rundungsfehler bearbeiten. Real-Zahlen kann er im Wertebereich von $3{,}4 \times 10^{-4932}$ bis $1{,}1 \times 10^{4932}$ bearbeiten.



◂ *Bild 1: Die Kalkulation eines Kalkulationsblatts kann mit einem 8087 erheblich schneller erfolgen.*

Führt ein eigens dafür ausgelegtes Programm eine mathematische Berechnung durch, so verwendet es den arithmetischen Coprozessor, statt den Mikroprozessor. Der Coprozessor führt die Berechnung durch und übergibt das Ergebnis dem Mikroprozessor. Die gesamte Operation benötigt nur einen Bruchteil der Zeit, die der Mikroprozessor benötigen würde. Um Ihnen einen Eindruck zu vermitteln, wie schnell der Coprozessor ist, vergleicht *Bild 1* die Neuberechnung einer Tabelle mit und ohne 8087-Coprozessor.

Neben der schnelleren Programmausführung, spart der Coprozessor auch Programmierzeit. Da die trigonometrischen, logarithmischen und exponentiellen Funktionen im Coprozessor hardwaremäßig vorhanden sind, braucht der Programmierer diese Funktionen nicht selbst zu schreiben. Mit diesen hardwaremäßigen Funktionen sind auch die Programme kleiner. Die Coprozessoren bieten Befehle für viele numerische Operationen wie die Zahlenkonvertierung, arithmetische Operationen und transzendente Funktionen (Tangens, Exponentialrechnung und Logarithmen).

Ein Coprozessor ist der billigste Weg, um die Verarbeitungsgeschwindigkeit von rechenintensiven Programmen zu erhöhen. Für den Bruchteil des Preises einer Beschleunigungskarte kann der Coprozessor Fließkommaoperationen beschleunigen. Außerdem benötigt er keinen (oder keine zwei) Erweiterungssteckplätze, da der Sockel für ihn schon auf der Hauptplatine vorhanden ist.

Es gibt drei verschiedene mathematische Coprozessoren: den 8087 (für den 8086 und den 8088), den 80287 (für den 80286) und den 80387 (für den 80386). Sowohl 8087, als auch 80287 gibt es in drei Geschwindigkeits Ausführungen. Die von Ihnen benötigte Ausführung hängt von der Taktfrequenz des Coprozessor-Sockels Ihres Computers, nicht von der Taktfrequenz des Mikroprozessors ab. Zum Beispiel versorgen einige 10-MHz-Computer den Sockel mit 8 MHz, und benötigen deshalb einen Coprozessor mit 8 MHz. Wenn Sie sich über die Geschwindigkeit des von Ihnen benötigten Coprozessors nicht sicher sind, so sollten Sie den Hersteller des Computers befragen.

Hunderte von Applikationen wurden so programmiert, daß sie Gebrauch von der Geschwindigkeit und der Präzision des Coprozessors machen. Diese sind sowohl aus dem kaufmännischen Bereich und dem Ingenieurwesen sowie grafische Anwendungen und Statistikpakete.

▸ *Tabelle 1: Die Ausführungszeit für Fließkomma-Berechnungen verkürzt sich stark, wenn ein 8-MHz-IBM-PC mit einem Coprozessor ausgerüstet ist.*

Befehl	Ungefähre Ausführungszeit (in Mikrosekunden)	
	Mit 8087	Ohne 8087
Addition/Subtraktion	10,6	1.000,0
Multiplikation (kurze Fließkommazahlen)	11,9	1.000,0
Multiplikation (temporäre Fließkommazahlen)	16,9	1.312,5
Division	24,4	2.000,0
Vergleich	5,6	812,5
Laden (lange Fließkommazahlen)	6,3	1.062,5
Speichern (lange Fließkommazahlen)	13,1	750,0
Quadratwurzel	22,5	12.250,0
Tangens	56,3	8.125,0
Exponentiation	62,5	10.685,5

Auch viele Compiler und Assembler unterstützen den Coprozessor. Der Gebrauch eines Coprozessors mit all diesen Programmen könnte nicht einfacher sein, da die Schnittstelle zwischen Mikroprozessor und Coprozessor eingebaut ist. Der einzige Unterschied, den Sie bemerken werden, ist die erhöhte Verarbeitungsgeschwindigkeit.

Entwicklungswerkzeuge

Fast alle Compiler und Assembler erzeugen Coprozessor-Code. Dies gilt für alle derzeitigen Versionen von Microsoft C, Pascal und Fortran, ebenso wie für Borlands TurboPascal. Unabhängig von der verwendeten Sprache, ist die Bearbeitung komplizierter mathematischer Ausdrücke mit einem Coprozessor einfach.

In einer Hochsprache ist die Verwendung eines Coprozessors völlig problemlos. Die Coprozessor-Befehle, wie Sinus, Wurzel, Tangens Hyberbolicus und Logarithmen werden in den von den Herstellern gelieferten Bibliotheken verwendet.

Assembler-Programmierer, die den Microsoft Macro Assembler der Version 1.25 oder höher verwenden, können entweder Programme schreiben, die direkt den Coprozessor ansprechen, oder Sie können diesen indirekt ansprechen, indem Sie auf Bibliotheken von Microsoft C, Pascal oder Fortran zurückgreifen. Zusätzlich kann eine Anzahl von Bibliotheken unabhängiger Softwarehersteller verwendet werden, die sich auf Mathematikbibliotheken spezialisiert haben.

Obwohl die meisten der Programme Bibliotheksfunktionen mit oder ohne Coprozessor aufrufen können, laufen diese Programme auf einem Computer mit Coprozessor wesentlich schneller. *Tabelle 1* zeigt die Erhöhung der Ausführungsgeschwindigkeit von Fließkommaoperationen eines typischen Tabellenkalkulationsblatts auf einem 8-MHz-Computer mit installiertem Coprozessor.

Viele Hochsprachen linken eine Emulationsbibliothek in jedes Programm, das Fließkommazahlen-Berechnungen durchführt. Es wird außerdem Code erzeugt, der das Vorhandensein des Coprozessors zur Laufzeit testet. Ist dieser vorhanden, so wird er benutzt. Ist er nicht vorhanden, so wird die Emulationsbibliothek benutzt. So können Programme, die geschrieben wurden, um den Coprozessor auszunutzen, auch auf einem Computer laufen, der keinen Coprozessor installiert hat.

Die Fehlersuche in Programmen mit Coprozessor-Befehlen ist nicht viel schwieriger als in Programmen ohne diese Befehle. Ein guter Debugger, so wie etwa CodeView, der im Microsoft C-Compiler seit der Version 4.0 enthalten ist, gestattet Ihnen das Ansehen und Ändern sowohl der Coprozessor-Register, als auch der Kontroll- und Statusregister. CodeView zeigt die Datenregister sowohl in ihrer internen hexadezimalen 80-Bit-Darstellung als auch ihren dezimalen Wert an. So ist die Fehlersuche in Fließkommabefehlen nicht viel schwieriger als in Mikroprozessorbefehlen.

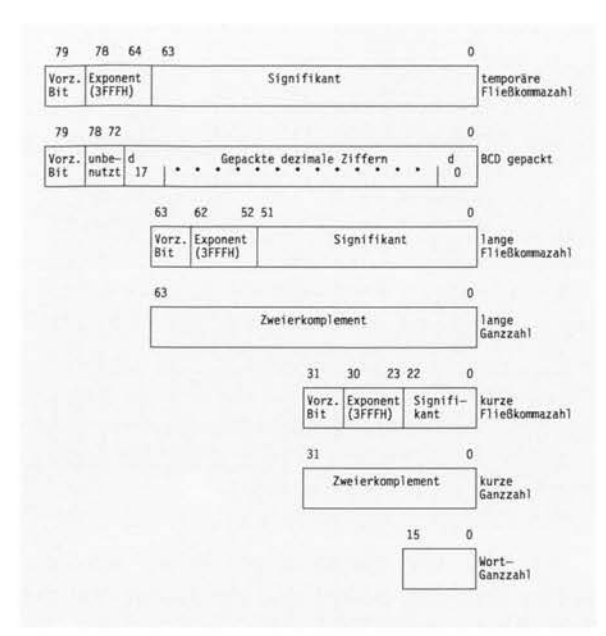
Synergy

Der Coprozessor ist eine Erweiterung des Mikroprozessors (Intel bezeichnet den Coprozessor auch als numerische Prozessorerweiterung). Sie benützen dieselben Busse und Speicher. Die Statusleitungen des Mikroprozessors sind direkt mit dem Coprozessor verbunden, wodurch der Coprozessor die Befehlsfolge des Mikroprozessors verfolgen kann. Der Coprozessor verfolgt und dekodiert die Befehle ohne jeglichen Zusatzaufwand. Er liest jeden Befehl in seine Befehlsschlange, führt aber nur Befehle aus, die für ihn bestimmt sind und behandelt die Mikroprozessorbefehle wie NOPs (keine Aktion). Im Gegensatz dazu behandelt der Mikroprozessor die Anweisungen des Coprozessors wie NOP's, und führt seinerseits nur die für ihn bestimmten Anweisungen durch. Der Mikroprozessor kontrolliert die Programmausführung, der numerische Coprozessor die numerischen Operationen.

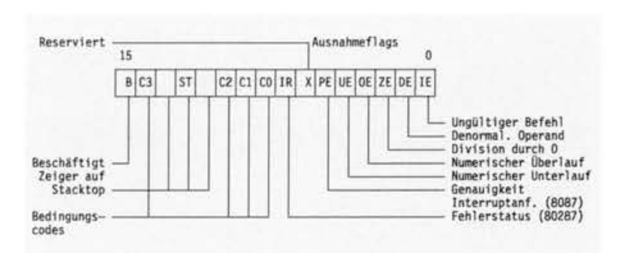
Anstelle der 8-Bit-Register des 8088, der 16-Bit-Register des 80286 oder der 32-Bit-Register des 80386, verfügt der Coprozessor über 80-Bit-Datenregister, wodurch er mehr Informationen speichern kann. Die Register des Coprozessors sind für einen speziellen Datentyp ausgelegt, und unterscheiden sich stark von den Allzweckregistern des Mikroprozessors. Die beiden Bausteine können aber trotzdem Daten über gemeinsame Speicherbereiche austauschen.

Datentypen

Die Register des Coprozessors sind für die Aufnahme von 80-Bit-Fließkommazahlen ausgelegt. Diese Zahlendarstellung, die Intel »temporary real« bezeichnet, ist identisch mit dem vorge-



◄◄ *Bild 2:* *Der Coprozessor kann mit sieben numerischen Formaten arbeiten, die maximal 80 Bit lang sind.*



◄ *Bild 3:* *Das 16 Bit breite Statusregister des Coprozessors dient als Flagregister.*

schlagenen IEEE 754 Fließkomma-Standardformat. Diese Zahl besteht aus einem Vorzeichenbit, einem 15 Bit breiten Exponenten und einem 64 Bit breiten Signifikanten. Obwohl der Coprozessor alle Daten in diesem Format speichert, kann er sechs andere Formate lesen und schreiben. Diese Formate sind: dezimal gepackt, lange Fließkommazahlen, lange Ganzzahlen, kurze Fließkommazahlen, kurze Ganzzahlen und 2-Byte-Ganzzahlen (*Bild 2*). Die Coprozessor-Speicher- und Ladebefehle wandeln automatisch die anderen sechs Datenformate in das temporäre Fließkommaformat und zurück. Der Microsoft Macro Assembler erlaubt die Deklaration dieser Formate mit den Anweisungen DW (2-Byte-Ganzzahlen), DD (kurze Ganzzahlen und kurze Fließkommazahlen), DQ (lange Ganzzahlen und lange Fließkommazahlen) und DT (gepackte Dezimalzahlen oder temporäre Fließkommazahlen).

Der Coprozessor speichert die Zahlen im normalisierten Format (wissenschaftliche Darstellung). Eine Zahl ist normalisiert, wenn das Bit 63 des Signifikanten 1 ist. Der Coprozessor geht davon aus, daß die Zahl im Signifikanten eine Realzahl zwischen 1 und 2 ist. Das Exponentenfeld gibt die Zahl der Schiebeoperationen an, die der Signifikant geschoben werden muß, um die Originalzahl zu erhalten. Da der Exponent als vorzeichenlose Zahl dargestellt ist, muß ein Offset addiert werden, um negative Zahlen darstellen zu können. Dies gestattet dem Coprozessor den Vergleich zweier Zahlen, ohne zuerst die Exponenten zu berechnen, und spart daher Ausführungszeit.

Register

Die Coprozessor-Bearbeitung spielt sich in acht Registern ab. Auf die Register kann in LIFO-Stackorganisation zugegriffen werden. Die Operationen arbeiten mit den obersten beiden Registern. Die Register können aber auch als festes Registerset verwendet werden, indem den Operationen zwei Registeroperanden angegeben werden.

Im Gegensatz zu den Registern des Mikroprozessors haben die Register des Coprozessors keine eindeutigen Namen. Sie werden als indizierte Einträge im Stack behandelt. Das oberste Register ist mit ST(0), das nächste mit ST(1) und so weiter bezeichnet. Die Zahlen werden durch Ablegen auf den Stack in den Coprozessor geladen, und durch Wegnehmen vom Stack erhält man die Ergebnisse wieder. Viele der Coprozessor-Befehle arbeiten nur mit dem obersten Stackelement. Die meisten verwenden voreingestellt das oberste Stackelement. Alle Registeradressen sind relativ zum obersten Stackelement.

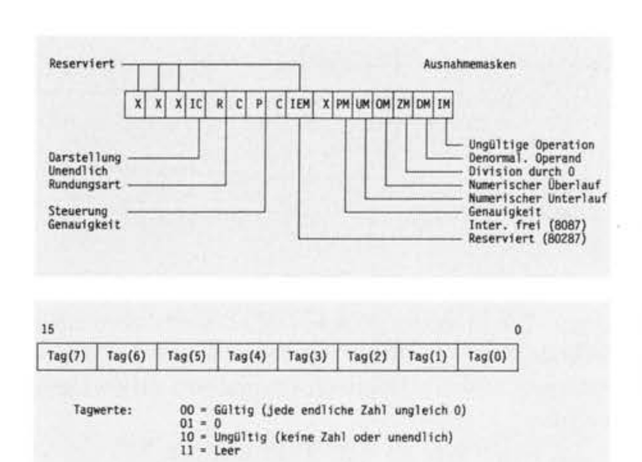
Ein 3 Bit großer Zeiger in einem anderen Register (Statuswort) zeigt auf das augenblickliche oberste Stackelement. Ein Push-Befehl erniedrigt diesen Zeiger um 1 und lädt einen Wert in das neue oberste Stackelement. Ein Pop-Befehl erhöht den Wert im Zeiger um 1 und entfernt das augenblicklich oberste Element. Der Stack ist zirkular und kann überschrieben werden, wenn er nicht richtig verwaltet wird.

Alle numerischen Befehle des Coprozessors benützen im Gegensatz zu den Kontrollbefehlen das oberste Stackelement als einen Operanden. Einige Befehle arbeiten nur mit dem obersten Stackelement, wogegen andere mit dem obersten und mit dem nächsten Stackelement arbeiten. Andere nehmen den zweiten Operanden von einem anderen Register des Stacks, oder aus dem Speicher.

Neben den acht Datenregistern verfügt der 8087 noch über fünf jeweils 16 Bit breite Register. Diese sind das Statuswort, das Kontrollwort, das Registerzustandswort (Tagwort), der Operandenzeiger und der Befehlszeiger.

Das Statuswort kann man sich als Flagregister vorstellen (*Bild 3*). Es enthält ein Flag, das gesetzt ist, wenn der Coprozessor beschäftigt ist, den Stackzeiger, Bedingungscodes und Ausnahmeanzeigen. Um dieses Statusregister mit Microsoft C zu lesen, können Sie die Funktion _status87 verwenden. Um es in einem Assemblerprogramm auszulesen, müssen Sie den Befehl FSTSW ausführen. Dieser Befehl überträgt das Statuswort in den Speicher, wo es der Mikroprozessor dann weiterbearbeiten kann.

Das Kontrollwort beschreibt, wie der Coprozessor auf Ausnahmen reagieren soll (*Bild 4*). Es beschreibt auch die Präzision, wie das Ergebnis



► *Bild 4: Das Kontrollregister beschreibt die Reaktionen des Coprozessors auf Ausnahmebedingungen.*

15 ... 0

Tag(7)	Tag(6)	Tag(5)	Tag(4)	Tag(3)	Tag(2)	Tag(1)	Tag(0)

Tagwerte:
00 = Gültig (jede endliche Zahl ungleich 0)
01 = 0
10 = Ungültig (keine Zahl oder unendlich)
11 = Leer

► *Bild 5: Das Tagwort beinhaltet Informationen über die Werte jedes Datenregisters.*

COPROZESSOR STATUS — COPROZESSOR UMGEBUNG

Real Mode	Protected Mode
Kontrollwort	Kontrollwort
Statuswort	Statuswort
Tagwort	Tagwort
Befehlsadresse	Befehlsoffset
Ad.19–16 | 0 | Ad.10–0	Befehlsselektor
Operandenadresse(15–0)	Operandenoffset
Op.(19–16) | 0	Operandenselektor
ST(0)	ST(0)
ST(1)	ST(1)
ST(2)	ST(2)
ST(3)	ST(3)
ST(4)	ST(4)
ST(5)	ST(5)
ST(6)	ST(6)
ST(7)	ST(7)

►► *Bild 6: Das Environment des Coprozessors beinhaltet alle Register außer den Datenregistern. Der Status enthält alle Register.*

gerundet werden soll, und ob Unendlich vorzeichenlos oder mit Vorzeichen behandelt werden soll. Das Kontrollwort verfügt über drei Kontrollfelder und sechs Ausnahmemasken. Das Maskieren der Ausnahmebits veranlaßt den Coprozessor, alle Ereignisse dieser Art zu behandeln. Sind sie nicht maskiert, so hat der Programmierer dafür Sorge zu tragen, daß die Ausnahmen richtig behandelt werden.

In Assembler wird das Kontrollwort in eine Speicherzelle geschrieben und von dort mit einem Befehl in den Coprozessor geladen. Programmierer, die in einer Hochsprache arbeiten, sollten Ihre Bibliotheksfunktionen ansehen, um herauszufinden, wie sie vorgehen müssen. Für Programmierer, die dieses Kontrollregister nicht verändern wollen, bietet Intel Standardeinstellungen. Hierbei sind Ausnahmen maskiert: 64 Bit Präzision, Rundung zur nächsten Zahl und projektive Unendlichkeit.

Das Tagwort enthält Informationen über den Inhalt jedes einzelnen Datenregisters (*Bild 5*). Diese Informationen werden hauptsächlich vom Coprozessor benutzt, um die Performance zu erhöhen. Der Coprozessor speichert 2 Bit für jedes Datenregister, was vier verschiedene Tagwerte ergibt.

Der Coprozessor braucht diese Tagwerte, um zu identifizieren, was in den einzelnen Datenregistern enthalten ist, und um ungültige Ergebnisse zu kennzeichnen. Der Coprozessor braucht das Tagwort auch, um die Richtigkeit der Stackinformation zu gewährleisten. Ist zum Beispiel ein Register als leer gekennzeichnet (Tagwert ist 11) und wird vom Stack genommen, so stellt der Coprozessor Stackunterlauf fest. Ebenso benützt der Coprozessor das Tagwort um Stacküberlauf zu entdecken, wenn ein Wert in einem Register abgelegt werden soll, das schon belegt ist. Stackunter- und -überlauf löst eine Ausnahme aus. Der Programmierer kann diese Ausnahme maskieren oder nicht maskieren (maskiert ist voreingestellt). Wenn entweder Stackunter- oder -überlauf auftritt und die ungültige Befehlsausnahme maskiert ist, setzt der Coprozessor den Stackpointer und gibt ein Standardergebnis zurück, das angibt, daß der Wert bedeutungslos ist.

Operand- und Befehlszeiger liefern Informationen über den Befehl und die Daten, die die Ausnahme veranlaßt haben, und werden für Fehlerbehandlungsroutinen benötigt. Die meisten Programmierer benötigen diese Register nicht, da Sie den Coprozessor die Ausnahmen selbst behandeln lassen.

Im Gegensatz zu den Statuswort- und Kontrollwortregistern kann auf das Tagwort, den Befehls- und den Datenzeiger nicht direkt zugegriffen werden. Auf diese Register kann indirekt zugegriffen werden, indem entweder das Environment (FSTENV) oder der Status des Coprozessors (FSAVE) in den Speicher geladen wird. Das 14 Byte große Environment des Coprozessors beinhaltet das Statuswort, das Kontrollwort, das Tagwort, den Befehls- und den Operandenzeiger. Der 94 Byte große Status des Coprozessors enthält das Environment und die acht Datenregister (*Bild 6*). Das Format des Status und des Environment des Coprozessors hängt vom Modus ab.

Tritt eine Ausnahme ein, während der Coprozessor im Realmodus ist, so liefert er die 20-Bit-Adresse des Befehls und des Operanden (falls vorhanden), und die niederwertigen 11 Bits des Opcodes. Im geschützten Modus (nur beim 80287 und beim 80387 möglich) liefert der Coprozessor den Selektor und den Offset des Befehls und des Operanden (falls vorhanden). Obwohl die 80287/80387 Realmodus-Ausnahmezeiger dasselbe Format haben wie beim 8087, zeigt der Befehlszeiger beim 80287/80387 auf den Prefix vor dem Opcode. Beim 8087 zeigt der Befehlszeiger auf die ESC-Anweisung selbst.

Ausnahmen

Der Coprozessor kennt sechs Ausnahmebedingungen: ungültiger Befehl, denormalisierter Operand, Division durch Null, numerischer Unterlauf, numerischer Überlauf und ungenaues Ergebnis. Die Ausnahmemasken des Coprozessors gestatten es dem Programmierer, die Ausnahmen selbst zu behandeln oder den Coprozessor einen festen Wert zurückgeben zu lassen. Tritt wäh-

```
title   umfang

.287                    ; MASM über das Vorhandensein von Coprozessor-
                        ; befehlen informieren.

data    segment
        radius  DD      2.468
        umfang  DD      ?
data ends

code    segment
        assume cs:code, ds:data
start:
        mov     ax,data ; Datensegment initialisieren
        mov     ds,ax
        finit           ; Coprozessor initialisieren
        fldpi           ; ST(0) = pi
        fadd    st,st   ; ST(0) = 2pi
        fld     radius  ; ST(1) = 2pi, ST(0) = radius
        fmul    st,st(1)        ; ST(0) = radius * 2pi
        fstp    umfang  ; Ergebnis ablegen und POP ST
        fwait           ; Ablage abwarten
        mov     ax,4c00h ; Zurück zu DOS
        int     21h
code    ends
        end     start
```

```
title   wurzel

.287                    ; MASM über das Vorhandensein von Coprozessor-
                        ; befehlen informieren.

bcd_data        segment
        array1  DT      1234567890,82,76543,8653
                DT      23456,8765,234567,875
                DT      5646456,23232,798778,43
        array2  DT 12 DUP (?)   ;Ergebnisse
bcd_data ends

code    segment
        assume cs:code, ds:bcd_data
start:
        mov     ax,bcd_data     ; Datensegment initialisieren
        mov     ds,ax
        finit                   ; Coprozessor initialisieren

        mov     cx,length array2; Schleifenzähler setzen
        mov     si,0            ; Index setzen

process_array:
        fbld    array1[si]      ; ST(0) = array1[index]
        fsqrt                   ; ST(0) = √ ST(0)
        frndint                 ; ST(0) auf Integer runden
        fbstp   array2[si]      ; Ergebnis im zweiten Array ablegen
        add     si,10           ; Index auf nächste zu radizierende
                                ; Zahl setzen
        loop    process_array   ; Solange bis CX <= Länge( array1 )

exit:
        fwait                   ; Letzte Ablage abwarten
        mov ax,4c00h            ; Zurück zu DOS
        int 21h
code    ends
        end     start
```

◄◄ *Listing 1: Dieses Assemblerprogramm berechnet mit dem Coprozessor den Umfang eines Kreises mit gegebenem Radius. Alle Coprozessor-Befehle beginnen mit F.*

◄ *Listing 2: Dieses Programm berechnet mit dem Coprozessor die Quadratwurzel eines jeden Elements eines Arrays von BCD-Zahlen. Die Ergebnise werden in einem anderen BCD-Array abgelegt, und können so leicht für die Ausgabe konvertiert werden.*

rend der Befehlsausführung im Coprozessor eine Ausnahme auf, so setzt der Coprozessor das entsprechende Bit im Statusregister. Der Coprozessor prüft dann sein Kontrollregister, um zu sehen ob diese Ausnahme maskiert ist. Ist die Ausnahme maskiert, so benützt der Coprozessor seine eigene Logik, um ein Ergebnis zu ermitteln. Die Ausnahmebits im Statusregister behalten ihre Werte, bis sie explizit mit der FINIT oder FCLEX gelöscht werden. Der Programmierer braucht bei maskierten Ausnahmen nicht nach jeder Anweisung das Statusregister abprüfen. Das periodische Prüfen gewährleistet genaue Ergebnisse.

Die andere Methode ist das Demaskieren eines oder mehrerer Ausnahmebits und das Löschen des Interrupt Enable Bits des Coprozessors. Unter diesen Umständen löst eine Ausnahme einen Interrupt aus. Der Programmierer muß dann einen Interrupt Handler schreiben, der auf diese Ereignisse reagiert. Der Coprozessor beinhaltet eine Menge eingebauter Untersützung für das Schreiben solcher Routinen.

Befehle

Die Coprozessor-Befehle gliedern sich in sechs Bereiche: Datentransfer, Laden von Konstanten, transzendente Kalkulationen, Vergleiche, Arithmetik und Prozessorkontrolle. Eine Coprozessor-Anweisung kann in Assembler auf zwei verschiedene Arten programmiert werden. Als eine Mikroprozessor-ESC-Anweisung, der eine Zahl folgt (zum Beispiel ESC 0BH), oder durch ein Coprozessor-Mnemonik (FSTP). Alle Versionen des Microsoft Macro Assembler seit der Version 1.25 unterstützen diese Coprozessor-Mnemonics. ESC-Anweisungen werden nur für ältere Assembler benötigt, die aber nur noch selten benutzt werden. Programmierer, die in Hochsprachen arbeiten, brauchen sich um dies nicht zu kümmern, da der Compiler es für sie erledigt.

Ein Coprozessor-Befehl beginnt mit F. Die *Listings 1* und *2* zeigen Beispiele solcher Befehle in Assembler. *Listing 3* zeigt ein Programm, das vom Microsoft C-Compiler der Version 4.0 erzeugt wurde. Obwohl die Coprozessor-Anweisungen nicht in der Quelldatei vorhanden sind, so können Sie diese doch sehen, wenn Sie das Programm mit der Option /Fc übersetzen und in die .COD Datei sehen.

Beginnt eine Anweisung mit 11011, so erkennt sie der Mikroprozessor als Coprozessor-Anweisung und reagiert mit dem Bilden aller notwendigen Operanden und Adressen, die er auf dem Datenbus ausgibt. Den Rest der Anweisung ignoriert er. Der Mikroprozessor holt dann weitere Befehle und führt sie aus, bis er angewiesen wird, auf den Coprozessor zu warten.

Da der Mikroprozessor und der Coprozessor verschiedene Aufgaben zur gleichen Zeit erledigen können, können sie sich gegenseitig Daten überschreiben, oder Befehle versäumen, falls sie nicht synchronisiert arbeiten. Alle Hochsprachen synchronisieren die Aktivitäten automatisch. Assembler-Programmierer müssen dies selbst erledigen. Im Gegenzug für die erhöhte Arbeit, sind Assembler-Programmierer flexibler, und können einen höheren Durchsatz erzielen, falls Sie die Synchronisation sorgfältig durchführen.

Die 80286- und 80287-Prozessoren haben im Gegensatz zu den 8088- und 8086-Prozessoren eine eingebaute Synchronisation. Deshalb müssen die Programmierer konsequent FWAIT-Befehle nach der Speicheroperation des Coprozessors einsetzen.

Verwenden Sie ESC-Anweisungen, so müssen Sie eine FWAIT-Anweisung codieren, falls der Mikroprozessor auf Daten des Coprozessors wartet. Alle F..-Anweisungen besitzen eine FWAIT-Anweisung als erstes Byte. In diesem Fall bleibt Ihnen die Arbeit erspart. (Einige Coprozessor Befehle existieren in der FN..-Form, die den Assembler keine FWAIT-Anweisung erzeugen läßt).

Zur Synchronisation der Daten müssen bei 8088 und 8086 die 8087-Befehle synchronisiert werden. Da der Coprozessor die Befehle durch das Mitverfolgen beim Laden der Befehlsschlange des Mikroprozessors erhält, kann der 8087 einen Befehl verlieren, während er arbeitet und der 8088/8086 weiter Befehle holt und ausführt.

► *Listing 3: Dieses Programm zeichnet einen Kreis auf dem Bildschirm eines Computers mit Grafikkarte. Die Microsoft C-Bibliothek wird zum Berechnen der Sinus- und Cosinuswerte benützt. Die Coprozessor-Anweisungen erscheinen erst nach der Übersetzung.*

```
#include        "stdio.h"
#include        "math.h"

extern  set_graph_mode();
extern  set_text_mode();
extren  plot_point();

#define VERTICAL_CENTER 99.5
#define HORIZONTAL_CENTER 319.5
#define PI 3.1415927

main()
    {
    char ch;
    float radians, radius, aspect_ratio;

    aspect_ratio=2.1; /* Pixelformat dem Bildschirmformat anpassen */
    radius=90;

    set_graph_mode(); /* 640 x 200 Grafikmodus */

    /* Der Kreis soll aus einem Punkt pro 1/100 Grad Radian entstehen */
    for (radians=0; radians<2*PI; radians=radians+0.01)
        {
        long x,y;

        x=HORIZONTAL_CENTER+radius*aspect_ratio*cos(radians);
        y=VERTICAL_CENTER+radius*sin(radians);

        /* Pixel auf den Monitor bringen */
        plot_point((int)x,(int)y);
        }

    /* Auf Tastendruck warten */
    ch=getchar();

    /* Wieder in Textmodus zurückschalten */
    set_text_mode();
    }
```

Jedes Programm, das den Coprozessor verwendet, sollte aber auch ohne diesen laufen können. Bevor die Software den Coprozessor anspricht, sollte sie prüfen, ob er überhaupt vorhanden ist. Dies ist einfach durch die Initialisierung des Coprozessors und das anschließende Lesen des Kontrollworts zu erreichen. Ist ein Coprozessor vorhanden, so sind die Werte des Kontrollworts so gesetzt, wie Intel dies spezifiziert. Viele Bibliotheken verfügen bereits über diesen Test. Ist kein Coprozessor vorhanden, so sollte das Programm eine Emulationsbibliothek aufrufen, oder sich beenden. *Listing 4* zeigt hierfür ein Beispiel.

Real oder geschützt

Der 8087 arbeitet nur im Real-Modus. Die 80287 und 80387 können hingegen im Real- und im geschützten Modus arbeiten. Alle Programme, die für den 8087 geschrieben wurden, sind kompatibel zu denen für 80287/80387 im Real Modus. Die Ausführung der privilegierten Anweisung SETPM versetzt den 80287 oder den 80387 in den geschützten Modus. Sie können nur durch einen Hardware-Reset in den Real Modus zurückversetzt werden.

Der Arbeitsmodus des Mikroprozessors beeinflußt den Coprozessor auf zwei Arten: Ausnahmebehandlungen und Speicherzugriff. Das Speicherabbild des Befehls- und Datenzeigers nach einer FSTENV- oder FSAVE-Anweisung hängt vom Arbeitsmodus des Coprozessors ab (*Listings 1 und 2*). Jedes Programm, das diese Informationen auswertet, muß den Arbeitsmodus berücksichtigen, um richtige Ergebnisse zu liefern. Im geschützten Modus ist der Interrupt 16 für die Ausnahmebehandlung des Coprozessors zuständig. Coprozessor-Anweisungen die zu einer Ausnahme führen, lösen einen Interrupt 16 aus, falls die Ausnahme nicht maskiert ist. Der geschützte Modus verfügt auch über eine Regelung, wenn der Coprozessor nicht vorhanden ist (oder emuliert werden soll). Ein Interrupt 7 wird ausgelöst, wenn eine ESC-Anweisung zu bearbeiten ist, und das Emulationsbit (EM) im Statuswort des Mikroprozessors gesetzt ist. Dieser Trap-Mechanismus hilft dem Programmierer, systematisch Emulationscode in das Programm aufzunehmen.

MS OS/2 bietet grundlegende Ausnahmebehandlung für den Coprozessor, indem es die Ausnahmebehandlung des 80287 oder 80387 unterstützt. Es verfügt über keine Standard-Emulationsbibliothek für den Coprozessor. Diese muß vom Compilerhersteller verfügbar sein.

Im geschützten Modus prüft der Mikroprozessor alle Speicherzugriffe (inklusive derjenigen des Coprozessors) auf Speicherschutz-Verletzungen. Coprozessor-Applikationen, die im geschützten Modus laufen, müssen sich an diese Speicherzugriffsregeln halten. Jeder Verstoß verursacht entweder Interrupt 13 (wenn der Verstoß beim ersten Wort des numerischen Operanden auftritt) oder Interrupt 9 (wenn der Verstoß bei einem folgenden Wort auftritt).

Wenn Sie ein 8087-Programm auf ein System im geschützten Modus übertragen wollen, so müssen Sie mit einem 80286/80386-Assembler neu übersetzen. Dieser entfernt die überflüssigen FWAIT-Befehle und erzeugt einen kompakteren Code. Sie sollten auch die folgenden Änderungen durchführen:

• Entfernen Sie die Interruptkontroller-Anweisungen im Ausnahmebehandlungsteil des Programms.

• Streichen Sie die 8087-Anweisungen FENI/FNENI (Interrupts zulassen) und FDISI/FNDISI (Interrupts sperren). Der 80287 und 80387 ignoriert diese Anweisungen. Der Status des 80287/80387 wird nicht aktualisiert.

• Vergewissern Sie sich, daß der Interruptvektor 16 zu einer numerischen Ausnahmebehandlungsroutine zeigt.

• Schreiben Sie eine Mikroprozessor-Ausnahmebehandlungsroutine für Interrupt 7. Diese wird bei der Bearbeitung einer Coprozessor-Anweisung aufgerufen, wenn das Statuswort des Mikroprozessors TS=1 (Task umgeschaltet) oder EM=1 (Emulation) enthält.

• Schreiben Sie eine Ausnahmebehandlungsroutine für den Interrupt 9 (die aufgerufen wird, wenn das zweite oder weitere Wort eines Fließkommaoperanden außerhalb des Segments liegt) und den Interrupt 13 (die ausgelöst wird, wenn das erste Wort eines numerischen Operanden außerhalb des Segmentes liegt).

Marion Hansen, Lori Sargent

Listing 4: Diese Routine führt dieselben Multiplikationsfunktionen mit und ohne Coprozessor durch. Das Programm prüft zuerst das Vorhandensein des Coprozessors. Ist dieser vorhanden, so wird die Funktion imul_32 aufgerufen, ansonsten wird zur Funktion emulate_imul_32 verzweigt.

```
title   math_module

.287                    ; MASM über das Vorhandensein von Coprozessor-
                        ; befehlen informieren.

public  init_math
public  imul_32

        present equ    0
        missing equ    1

code    segment         public  'code'
        assume cs:code

        cp_flag         db      1       ; Status für '87er vor-
                                        ; handen / nicht vorhanden
        ctrl_word       dw      0       ; '87er Control-Wort

;-------------------------------------------------------------------;
; init_math: Sucht nach dem Coprozessor und setzt entsprechend dem
;            Ergebnis das globale Flag cp_flag. Es gibt an, ob
;            der Coprozessor oder die Emulation benutzt werden
;            soll. Diese Routine muß daher unbedingt VOR Aufruf
;            der anderen beiden gestartet werden.
;-------------------------------------------------------------------;

init_math       PROC    FAR

        fninit                          ; '87er initialisieren
        fnstcw  cs:ctrl_word            ; Controlwort ablegen
        cmp     byte ptr cs:[ctrl_word+1],3; Sind nur die Bits 8 und 9
        je      yes_cp                  ; gesetzt, ist ein Co-
        mov     cs:cp_flag,missing      ; prozessor installiert
        jmp     init_math_exit

yes_cp:
        mov     cs:cp_flag,present

init_math_exit:
        ret

init_math       ENDP


;-------------------------------------------------------------------;
; imul_32: Diese Routine multipliziert zwei 32-Bit-Integer mit Vor-
;          zeichen. Sie kann auch für die Multiplikation von zwei
;          32-Bit-Festkommazahlen genutzt werden.
;
;          Input:  Zwei 32-Bit-Integer
;                  ds:si Zeiger auf Integer A
;                  ds:di Zeiger auf Integer B
;
;          Output: 64-Bit-Ergebnis, abgelegt ab Adresse [es:bx]
;-------------------------------------------------------------------;

imul_32         PROC    FAR

        cmp     cs:cp_flag,missing      ; Falls kein Coprozessor
        je      emulate_imul_32         ; vorhanden, dann Emulation
                                        ; sonst 80X87 benutzen.
        fild    dword ptr [si]          ; ST(0)=A
        fimul   dword ptr [di]          ; ST(0)=A*B
        fistp   qword ptr es:[bx]               ; Ergebnis ablegen und POP
        fwait                           ; Ablage abwarten
        jmp     imul_32_exit            ; Fertig

;-------------------------------------------------------------------;
; Bei der Emulation teilen sich A und B wie folgt auf:
;       A ist ein 32-Bit-Integer bestehend aus
;         einem low word A0 und einem high word A1 und
;       B ist ein 32-Bit-Integer bestehend aus
;         einem low word B0 und einem high word B1.
;
; Das Ergebnis wird durch Addition der vier Teilprodukte berechnet.
; Diese Teilprodukte ergeben sich aus den vorzeichenlosen Multipli-
; kationen von A0*B0, A0*B1, A1*B0 und A1*B1. Das Vorzeichen des
; Endergebnisses wird am Schluß der Berechnungen korrigiert.
;-------------------------------------------------------------------;

emulate_imul_32:

        push    ax                      ; Zusätzliche Register sichern
        push    cx
        push    dx
        push    bp

A0_x_B0:
        mov     ax,[si]         ; ax=A0
        mul     word ptr [di]   ; dx=A0B0h, ax=A0B0l
        mov     es:[bx],ax      ; A0B0l ablegen (4.Spalte)
        mov     cx,dx           ; cx=A0B0h
A1_x_B0:
        mov     ax,[si+2]       ; ax=A1
        mul     word ptr [di]   ; dx=A1B0h, ax=A1B0l
        push    bx              ; Zeiger später wiederverwenden
        mov     bx,ax           ; bx=A1B0l
        mov     bp,dx           ; bp=A1B0h
A0_x_B1:
        mov     ax,[si]         ; ax=A0
        mul     word ptr [di+2] ; dx=A0B1h, ax=A0B1l
        add     cx,bx           ; cx=A0B0h+A1B1l
        adc     cx,ax           ; cx=A0B0h+A1B1l+A0B1l+carry
        pop     bx              ; Ablagezeiger holen
        mov     es:[bx+2],cx    ; 3.Spalte speichern
        push    bx              ; Zeiger später wiederverwenden
        xor     bx,bx           ; bx löschen
        adc     bx,0            ; Carry-Information sichern
        mov     cx,dx           ; cx=A0B1h
A1_x_B1:
        mov     ax,[si+2]       ; ax=A1
        mul     word ptr [di+2] ; dx=A1B1h, ax=A1B1l
        add     cx,bx           ; cx=A0B1h+carry
        adc     cx,bp           ; cx=A0B1h+A1B0h
        adc     cx,ax           ; cx=A0B1h+A1B0h+A1B1l+carry
        pop     bx              ; Ablagezeiger holen
        mov     es:[bx+4],cx    ; 2.Spalte speichern
        adc     dx,0            ; dx=A1B1+carry
        mov     es:[bx+6],dx    ; 1.Spalte speichern

; Jetzt muß noch eine Vorzeichenkorrektur durchgeführt werden.

test_A:
        mov     ah,[si+2]       ; ah=high byte von A
        or      ah,ah           ; Falls A negativ ist,
        js      substract_B     ; subtrahiere B vom high DD des Ergebnisses

test_B:
        mov     ah,[di+2]       ; ah=high byte von B
        or      ah,ah           ; Falls B negativ ist,
        js      substract_A     ; subtrahiere A vom high DD des Ergebnisses

        jmp     emulate_done

substract_B:
        mov     ax,[di]         ; ax=B0
        mov     cx,[di+2]       ; cx=B1
        sub     es:[bx+4],ax    ; die beiden high words abgleichen
        sbb     es:[bx+6],cx
        jmp     test_B

substract_A:
        mov     ax,[si]         ; ax=A0
        mov     cx,[si+2]       ; cx=B1
        sub     es:[bx+4],ax    ; die beiden high words abgleichen
        sbb     es:[bx+6],cx

emulate_done:
        pop     bp              ; Register wiederherstellen
        pop     dx
        pop     cx
        pop     ax

imul_32_exit:
        ret

imul_32 ENDP

code    ends
        end
```

Impressum Microsoft System Journal

Erscheinungsweise
Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats (ISSN 0933-9434).
Den Heften liegt eine Diskette bei.

Herausgeber
Microsoft GmbH
Edisonstr. 1, D-8044 Unterschleißheim
Tel.: 089 / 31705-0
Telefax: 089 / 3 17 05-400
Teletex/Telex: (17) 89 83 28 MS-GmbH

Redaktion
Synergy Verlag GmbH
Redaktion *Microsoft System Journal*
Theresienstraße 40, D-8000 München 2
Tel.: 089 / 28 06 85, Telefax: 089 / 28 21 63
Günter Jürgensmeier, Hartmut Niemeier

Mitarbeiter
Udo Borkowski, Michael Bülow, Marcellus Buchheit, Greg Comeau, Terry R. Dettmann, Susan Franklin, Marion Hansen, Günter Jürgensmeier, Michael Kausch, Rainer Kretzer, Thomas Langhammer, Dan Mick, Hartmut Niemeier, Tony Peters, Lori Sargent, Richard Hale Shaw, Franz-Josef Steiner, Nick Stuecklen, Michael Tischer, Wolfgang Wirth, Jochen Witte

Manuskripteinsendungen
Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.

Typographie
Rudolf Paulus Gorbach, D-8035 Gauting-Buchendorf

Titelbildentwurf
Hermann Menig, D-8421 Wildenberg

Anzeigen
Marketing Projekt 2000 GmbH
Gottfried-Böhm-Ring 59, D-8000 München 70
Tel.: 089 / 7 85 58 82, Telefax: 089 / 78 19 28

Druck
Alois Erdl KG, D-8223 Trostberg

Vertriebsleiter
Axel Herbschleb, Vogel Verlag und Druck KG
Postfach 6740, Max-Planck-Straße 7/9
8700 Würburg 1
Tel.: 0931 / 418-2198, Telefax: 0931 / 418-2120

Vertrieb Handelsauflage
Inland (Bahnhofsbuchhandel und ausgewählter Zeitschriftenhandel):
Vereinigte Motor-Verlage GmbH & Co. KG
Leuschnerstraße 1, 7000 Stuttgart 1
Tel.: 0711 / 2043-1, Telex: 7 22 036
Ausland:
Deutscher Pressevertrieb, Buch Hansa GmbH
Wendenstr. 27-29, 2000 Hamburg 1
Tel.: 040 / 23711-0, Telex: 2162401

Abbonement-Bestellungen
Vogel Verlag und Druck KG
Abonnement Service *Microsoft System Journal*
Postfach 6740, 8700 Würzburg 1
Tel.: 0931 / 418-2019
Telefax: 0931 / 418-2120, Udo Kleindienst

Bezugspreise
Jahresabbonnement Inland 126,- DM (117,76 DM + 8,24 DM Umsatzsteuer), Ausland:
Österreich: 890 öS, Schweiz: 126 sfr, übriges Ausland: 132 DM
Abonnementspreise inkl. Versandkosten. Einzelheftpreise siehe Titelseite. Einzelheftpreis zzgl. Versandkosten. Schüler- und Studenten-Jahresabonnements werden mit 30% rabattiert (nur gegen Nachweis). Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.

Bankverbindungen
Abonnement Inland und Ausland außer Österreich und Schweiz:
nur auf Postgiroamt, Stuttgart (BLZ 600 100 70) 2117 17-707
Einzelheft Inland:
nur auf Postgiroamt, Stuttgart (BLZ 600 100 70) 2385 25-705
Abonnement Schweiz:
nur auf Postscheckamt Basel, Kontonummer 40-37295-6, Referenznummer 22956

Urheberrecht
Copyright (C) 1989 Microsoft GmbH. Alle Rechte vorbehalten.
Alle im *Microsoft System Journal* erschienenen Beiträge und die Programme und Daten auf der beigelegten Diskette sind urheberrechtlich geschützt. Alle Rechte, auch Übersetzungen, vorbehalten. Reproduktionen gleich welcher Art, ob Fotokopie, Mikrofilm oder Erfassung in Datenverarbeitungsanlagen, nur mit schriftlicher Genehmigung der Microsoft GmbH. Anfragen sind an Michael Bülow zu richten.
Für die Programme, die als Beispiele veröffentlicht werden, kann der Herausgeber weder Gewähr noch irgendwelche Haftung übernehmen. Aus der Veröffentlichung kann nicht geschlossen werden, daß die beschriebenen Lösungen oder verwendeten Bezeichnungen frei von gewerblichen Schutzrechten sind. Die Erwähnung oder Beurteilung von Produkten stellt, soweit es sich nicht um Microsoft-Produkte handelt, keine irgendwie geartete Empfehlung der Microsoft GmbH dar. Für die mit Namen oder Signatur gekennzeichneten Beiträge übernimmt der Herausgeber lediglich die presserechtliche Verantwortung.

Herstellung
Das *Microsoft System Journal* wird mit Microsoft Word 5.0 geschrieben und gestaltet. Der Ausdruck erfolgt auf einer Linotronic 300.

Vorschau

Die Ausgabe 1/90 (Januar/Februar) erscheint am 20.12.89 mit folgenden Themen:

MS OS/2
Schnellere Grafiken unter dem Presentation Manager
Die Erstellung ansprechender Benutzeroberflächen
Aufbau und Einsatz von Dynamic Link Libraries
OS/2-Systemaufrufe von Basic aus verwenden

Microsoft C
Pointer in C verstehen und einsetzen
SAA-Serie, Teil 7
C-Kurs für Umsteiger, Teil 3

Windows
Windows-Dialogboxen mit neuen Steuerungsklassen
Einführung in die objektorientierte Programmierung mit Actor

MS-DOS
Die EXEC-Funktion von MS-DOS
Die Verwendung von Extended und Expanded Memory

Darüber hinaus berichten wir über einige neue und interessante Windows-Produkte von Microsoft und anderen Anbietern.

Die Microsoft Hotline

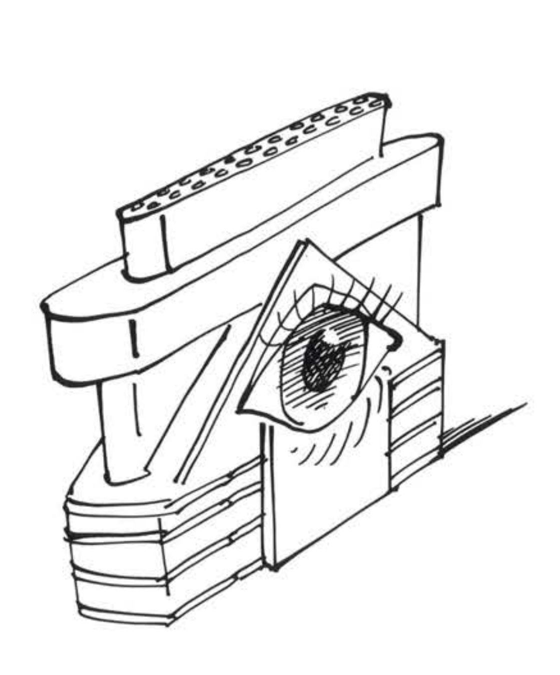
Die Microsoft-Hotline steht Montag bis Freitag zwischen 9 und 12 Uhr sowie zwischen 13 und 16 Uhr für telefonische Produktinformationen zur Verfügung. Für die verschiedenen Produkte können folgende Durchwahlnummern gewählt werden:

Telefonnr.	Produkte
089/31705-81	Microsoft Produkte für Macintosh
089/31705-82	Microsoft Multiplan, Microsoft Excel, Microsoft Chart, Microsoft Works
089/31705-83	Microsoft Word, Microsoft Windows Write, Word Druckertreiber
089/31705-84	Microsoft Flightsimulator, Microsoft Windows Utilities, Microsoft Paintbrush, EasyCad
089/31705-85	Microsoft Windows/286, Microsoft Windows/386, Microsoft Project, Microsoft Maus, Netzwerkapplikationen, Extended/Expanded Memory, LIM Standard
089/31705-86	Microsoft Compiler, Microsoft Quick Sprachen

Inserenten

AIT	92
Albrecht Software	160
BKS-Software	85
C.I.T.	101
CE Plus	87
3Com	84/85
ECO-Institut	85
ENZ	92
Fast Electronic	195
Format	177
HEC Hanseatische	135
Kickstein Software	92
Lauer & Wallwitz	29
Marketing Projekt 2000	101
Markt & Technik Buchverlag	48/49, 196
Microsoft	7, 58/59
M.I.S.	34
Müller Software	186
PEM	140
Schneider + Koch	79
SPI	21, 65
Synergy Verlag	48/49, 149
Systhema Verlag	111
te-wi Verlag	2
Vieweg	97
Vogel Verlag	111, 123
Zoschke	92

Liebe Softwareknacker. Es ist völlig zwecklos, dem neuen Hardlock E-Y-E schöne Augen zu machen.







Softwareschutz braucht einen seriösen Partner.
Wer beim Schutz seiner Software kein Auge mehr zudrücken möchte, landet irgendwann bei FAST Electronic. Seit 1985 haben wir über 120.000 Hardlock-Module produziert. Und verkauft. Denn Hardlock ist kein gewöhnlicher Kopierschutz. Hardlock ist Softwareschutz durch Hardware. Transparent. Anreihbar. Und durch die einzigartige Encryption-Technique nicht zu knacken.

Hardlock E-Y-E – ein eigener Chip für den Softwareschutz.
Der Vorsprung wird größer. Das neue Hardlock E-Y-E basiert auf einem Chip, den FAST Electronic eigens zum Schutz von Software entwickelt hat. Hardlock E-Y-E ist cryptoprogrammierbar. Das heißt, es kann von Ihnen selbst programmiert werden – und von niemand anderem. So sind Sie flexibler beim Schutz von Programmoptionen. Im neuen Hardlock E-Y-E können optional bis zu 128 Byte Daten abgelegt werden. Und mit dem automatischen Einbindungsprogramm HL-Crypt schützen Sie Ihre .COM- und .EXE-Files in Sekundenschnelle. Für die individuelle Einbindung erhalten Sie ohne Aufpreis Abfrageroutinen für alle gängigen Compiler.

Testen Sie den neuen Chip.
FAST Electronic hat eine halbe Million Deutsche Mark investiert, um Hardlock noch sicherer zu machen. Aber nicht teurer. Es lohnt sich, unsere Konditionen zu kennen. Und Leistung und Preis mit anderen Systemen zu vergleichen. Sie werden sehen: Gerade bei großen Stückzahlen können wir Ihnen konkurrenzlos günstige Angebote kalkulieren. Bestellen Sie ein Test-Modul des neuen Hardlock E-Y-E unverbindlich zur Ansicht. Jetzt.



Das neue Hardlock E-Y-E: Sicherheit in Silizium.



Fast Electronic GmbH

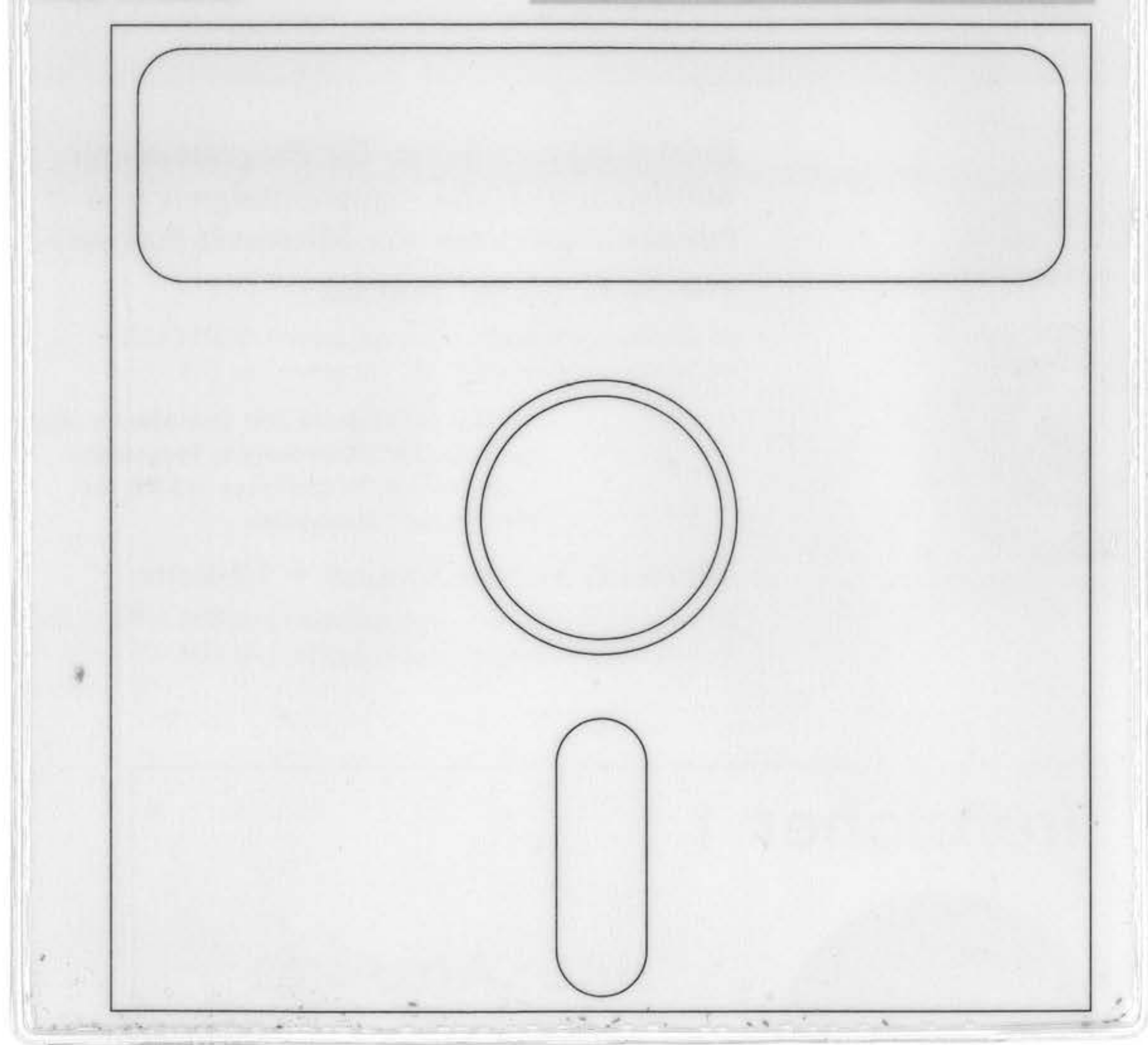
FAST Electronic GmbH, Kaiser-Ludwig-Platz 5, 8000 München 2, Tel. (089) 53 26 53, Fax (089) 53 34 01

Die Diskette zum Microsoft System Journal Nov./Dez. 1989

Verzeichnisstruktur der Diskette MSJDSK36

Verzeichnis	Dateien
\	README.EXE READ.ME
\8087	MATHMOD.ASM, UMFANG.ASM, WURZEL.ASM, GRAPH.C
\DDE	CLIENT.C, CLIENTMN.C, CLINIT.C, SERVER.C, SRVRINI.C SRVRMAIN.C, CLIENT.H,SERVER.H,ST.H
\DEBUG	DEBUGSRC.ASM
\DISKS	FREE.C, LIST1.C, LIST2.C, LIST3.C, LIST4.C, LIST6.C
\DUMP	DUMP.C, FILE.C, PANE.C, SPLITH.CUR, SPLITV.CUR, SPLITX.CUR DUMP.DFD, DUMP.DFP, DUMP.DLG, DEFS.H, DUMP.H, DUMP.ICO DUMP.MD, DUMP.MP, DUMP.RC
\OVER	OV, OVASM, SMERGE, OV.ASM, OVASM.ASM, OVRMGR.ASM P1ASM.ASM, P2ASM.ASM, RDOVLY.ASM, SUPPORT.ASM OV.C, P1.C, P2.C, RDOVLY.C, SMERGE.C, EXEHDR.H
\QP	DISKPARA.PAS
\SAA6	DLGDEMO.C, DLGPB.C, DLGRB.C, DLG.H
\THREADS	HELLOO, HELLOO.C

README.EXE und READ.ME:
Diese Datei und ein Programm zur Anzeige derselben.
\8087: Zu »Auf, auf und davon«, S. 187
\DDE: Zu »Dynamischer Datenaustausch unter dem Presentation Manager«, S. 6
\DEBUG: Zu »Die Debugregister des Intel 386«, S. 178
\DISKS: Zu »Diskettenstruktur unter DOS«, S. 150
\DUMP: Zu »Eine Hex-Dump-Applikation für Windows«, S. 35
\OVER: Zu »Ein Overlay-Manager für DOS«, S. 161
\QP: Zu »Systemnahe Programmierung mit Quick-Pascal«, S. 80
\SAA6: Zu »Noch mehr Objekte für Ihre Dialogboxen«, S. 115
\THREADS: Zu »Multithread-Programme unter OS/2«, S. 19



Sollte sich diese Diskette als defekt erweisen, schicken Sie sie bitte an die untenstehende Adresse. Sie erhalten dann umgehend ein neues Exemplar.

Microsoft System Journal
Vertriebsservice
Postfach 6740
D-8700 Würzburg 1

Allen Interessenten bieten wir die Listings gegen eine Aufwandsentschädigung von DM 8,– auch auf 3 1/2-Zoll-Disketten an. Wenden Sie sich dazu an:

Microsoft System Journal
Leserservice 7311
Postfach 6740
D-8700 Würzburg

Jetzt zum Abo-Vorteilspreis bestellen!



Vorsprung sichern · aus erster Hand informieren.
MS Journal ..\etc
im Jahresabonnement
nur DM 36,–
im 2-Jahresabonnement
nur DM 64,–



Jetzt zum Abo-Vorteilspreis bestellen!



Insider-Informationen für Programmierer, Softwareentwickler, Systemdesigner und erfahrene Anwender von Microsoft-Software.

Microsoft System Journal

im Jahresabonnement (6 Ausgaben) zu DM 115,–
im 2-Jahresabonnement (12 Ausgaben) zu DM 210,–

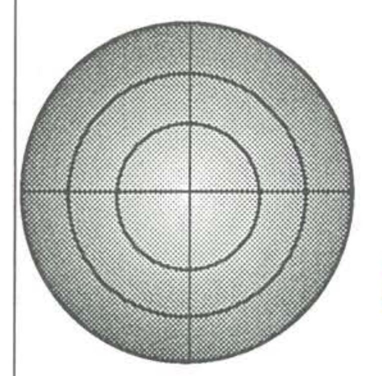


Listings auf Diskette zum Compilieren oder zur schnellen Übernahme in Programme. Lieferbar auf 5¼"Disketten 360 KB für IBM PC und Kompatible

Microsoft System Journal + Diskette

im Jahresabonnement (6 Ausgaben) zu DM 230,–
im 2-Jahresabonnement (12 Ausgaben) zu DM 420,–

Treffsicher !



Ihre Anzeige
im
Microsoft
SYSTEM JOURNAL

Nutzen Sie die qualifizierte Zielgruppe für Ihre Produkt- und Stellenanzeigen.

Informationen : MARKETING PROJEKT 2000 GmbH; Tel. 089/7 85 58 8 2
Fordern Sie unsere Media-Informationen mit beiliegender Antwortkarte an !

MS Journal ..\etc · BESTELLUNG

Ja, ich bestelle die MS Journal \etc. ab: ______________________

☐ **für mich/uns** ☐ **für umseitigen Empfänger**

Für: ☐ 1 Jahr (12 Ausgaben) zum Vorteilspreis von DM 36,–
☐ 2 Jahre (24 Ausgaben) zum Vorteilspreis von DM 64,–

Postzustellung frei Haus. Vertriebskosten und Mehrwertsteuer sind im Vorteilspreis enthalten.

____
(Name (bitte in Blockschrift)

____
Vorname

____
Straße und Hausnummer

____
PLZ und Ort

Dieses Angebot gilt für die Bundesrepublik Deutschland und West-Berlin.
Auslandspreise: Schweiz sfr 40,–, Österreich öS 280,– (für 1 Jahr)
Schweiz sfr 64,–, Österreich öS 500,– (für 2 Jahre)

Microsoft System Journal Bestellung

Ja, ich bestelle
☐ ... Heft(e) Microsoft System Journal
Ausgabe(n) vom: ______________________
____

Zum Einzelpreis von DM 24,80
Auslandspreise: Schweiz sfr 24,80
Österreich öS 200,–

____
Name (bitte in Blockschrift)

____
Vorname

____
Straße und Hausnummer

____
PLZ und Ort

Ja, ich bestelle das Microsoft System Journal ab: ______

Für:

☐ 1 Jahr (6 Ausgaben) zum Vorteilspreis von DM 126,–

Postzustellung frei Haus. Vertriebskosten und Mehrwertsteuer sind im Vorteilspreis enthalten.

Dieses Angebot gilt nur für die Bundesrepublik Deutschland und West-Berlin.

Auslandspreise:
Schweiz sfr 126,– Österreich öS 900,– für 1 Jahr (6 Ausgaben).

Das Abonnement verlängert sich automatisch um ein weiteres Jahr zu den dann gültigen Preisen, wenn es nicht acht Wochen vor Ablauf gekündigt wird.

Ich bezahle mein Abonnement:

☐ Sofort nach Erhalt der Rechnung.

☐ Durch Bankeinzug.

Bitte die Einzugsermächtigung auf der Rückseite ausfüllen.

Microsoft SYSTEM JOURNAL

Empfänger der Zeitschrift

Name (bitte in Blockschrift)

Vorname

Straße und Hausnummer

PLZ und Ort

Garantie

Mir ist bekannt, daß ich diese Bestellung innerhalb einer Woche bei der Bestelladresse widerrufen kann. Zur Wahrung der Frist genügt die rechtzeitige Absendung meines Widerrufsschreibens.

Ich bestätige dies durch meine zweite Unterschrift.

Datum/Unterschrift

Bitte freimachen

Antwort

Druck- und Verlagshaus
Alois Erdl KG

Postfach

D-8223 Trostberg

Einzugsermächtigung

Hiermit ermächtige ich Sie widerruflich, die von mir zu entrichtenden Zahlungen für bei Ihnen bestellte Artikel bei Fälligkeit zu Lasten meines

Kontos Nr.: __________

BLZ: __________

Geldinstitut __________

durch Lastschrift einzuziehen. Wenn mein Konto die erforderliche Deckung nicht aufweist, besteht seitens des kontoführenden Geldinstituts keine Verpflichtung zur Einlösung.

Datum/Unterschrift

Garantie

Mir ist bekannt, daß ich diese Bestellung innerhalb einer Woche bei der Bestelladresse widerrufen kann. Zur Wahrung der Frist genügt die rechtzeitige Absendung meines Widerrufsschreibens.

Ich bestätige dies durch meine zweite Unterschrift.

Datum/Unterschrift

Bitte freimachen

Antwort

System Journal
Leserservice 7311

Postfach 6740

D-8700 Würzburg

Absender: